Final Project

Development of a software architecture for virtual sensors in an autonomous small vessel for inland waterways.[1]

August 05, 2021

Jonas Mahler

[1]Revisited version without formal part and with reduced number of external figures.

Abstract

This project takes place in the context of the research project Model Scale Development Platform for Maneuver Automation/ Entwicklungsplattform im Modellmaßstab für Manöver-Automatisierung (ELLA) that is funded by the German Federal Ministry of Transport and Digital Infrastructure. In this project, automated manoeuvring of inland navigation vessels in confined spaces such as flood gates and harbours is investigated with a simulated virtual environment and a subsequent verification of the simulation results with a model scaled small vessel of an actual inland navigation vessel. The aim of this thesis is to develop a software architecture that allows for the integration of virtual sensors in the navigation subsystem of a guidance, navigation and control system of a small vessel in the inland navigation sector. In addition to the integration of virtual sensors, the architecture should be easily extendable towards the utilization of real sensors.

The system is realized within the framework supplied by ROS2 in conjunction with Node.js and a virtual environment that is set up in a game engine provided by Unity3d. As a first step a virtual environment is set up in Unity3d. A small vessel that is contained in the virtual environment is equipped with virtual sensors such as a Light Detection and Ranging (LIDAR) and an Intertial Measurement Unit (IMU). The sensors publish its data with customized messages via Node.js that serves as a transition, to ROS2. In ROS2 the distinction between the three subsystems of the Guidance, Navigation and Control (GNC) system is made and the sensor data is preprocessed by the navigation subsystem if necessary.

The results show that the chosen frameworks and algorithms allow for virtual data generation and subsequent data processing and visualization. It is demonstrated that the designed software architecture allows for the modular integration of both virtual elements and algorithms. Ways to extend the design are outlined as well as improvements related to the performance of the implemented modules.

Kurzfassung

Diese Projektarbeit findet im Rahmen des Forschungsvorhaben ELLA statt, das vom Bundesministerium für Verkehr und digitale Infrastruktur gefördert wird. In diesem Projekt wird das automatisierte Manövrieren von Binnenschiffen in engen Räumen wie Schleusen und Häfen mit einer simulierten virtuellen Umgebung und einer anschließenden Verifikation der Simulationsergebnisse mit einem maßstabsgetreuen Kleinfahrzeug eines realen Binnenschiffs untersucht. Ziel dieser Arbeit ist es, eine Softwarearchitektur zu entwickeln, welche die Integration virtueller Sensoren in das Navigationssubsystem eines Führungs-, Navigations- und Steuerungssystems eines Binnenschiffs ermöglicht. Neben der Integration virtueller Sensoren soll die Architektur auch für die Nutzung realer Sensoren erweiterbar sein. Realisiert wird das System innerhalb des von ROS2 bereitgestellten Frameworks in Verbindung mit Node.js und einer virtuellen Umgebung, die in einer von Unity3d Game-Engine umgesetzt wird.

In einem ersten Schritt wird eine virtuelle Umgebung in Unity3d erstellt. Ein Kleinfahrzeug, das sich in der virtuellen Umgebung befindet, ist mit virtuellen Sensoren wie einem LIDAR und einer IMU ausgestattet. Die Sensoren veröffentlichen ihre Daten mit Nachrichten über Node.js, das als Übergang dient, an ROS2. In ROS2 wird die Unterscheidung zwischen den drei Subsystemen des GNC-Systems vorgenommen und die Sensordaten werden bei Bedarf durch das Navigationssubsystem vorverarbeitet.

Die Ergebnisse zeigen, dass die gewählten Frameworks und Algorithmen eine virtuelle Datengenerierung und anschließende Datenverarbeitung und Visualisierung ermöglichen. Es wird gezeigt, dass die entworfene Software-Architektur eine modulare Integration sowohl von virtuellen Elementen als auch von Algorithmen ermöglicht. Möglichkeiten zur Erweiterung des Designs werden ebenso aufgezeigt wie Verbesserungen in Bezug auf die Leistung der implementierten Module.

Contents

USV	Unmanned Surface Vehicle
GNC	Guidance, Navigation and Control
LIDAR	Light Detection and Ranging
GNSS	Global Navigation Satellite System
IMU	Intertial Measurement Unit
MOOS-IvP	Mission Oriented Operating Suite-Interval Programming
PID	Proportional Integral Derivative
PLC	Programmable Logic Control
RADAR	Radio Detection and Ranging
LASER	Light Amplification by Stimulated Emission of Radiation
CFT	Clustering Feature Tree
CF	Clustering Feature
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
SNR	Signal Noise Ratio
CFAR	Constant False Alarm Rate
OS-CFAR	Ordered Statistics Constant False Alarm Rate
PDF	Probability Density Function
CUT	Cell Under Test
DBSCAN	Density Based Spatial Clustering of Applications with Noise
SONAR	Sound Navigation and Ranging
SDK	Software Development Kit
CARACAS	Control Architecture for Robotic Agent Command and Sensing
ROS	Robot Operating System
CAN	Controller Area Network
GPU	Graphics Processing Unit
API	Application Programming Interface
HMI	Human Machine Interface
FOV	Field of View
LAESSI	Guiding and Assistance Systems to Improve Safety of Navigation on Inland Waterways
RTK	Real Time Kinematics

CPU	Central Processing Unit
MASS	Maritime Autonomous Surface Ships
VDE	Association for Electrical, Electronic and Information Technologies
ISO	International Organization for Standardization
IMO	International Maritime Organization
IEEE	Institute of Electrical and Electronics Engineers
IEC	International Electrotechnical Commission
DDS	Data Distribution Service
RTT	Round Trip Time
TCP	Transmission Control Protocol
JSON	JavaScript Object Notation
ELLA	Model Scale Development Platform for Maneuver Automation/ Entwicklungsplattform im Modellmaßstab für Manöver-Automatisierung
ROS2	Roboter Operating System 2
TIN	Triangulated Irregular Network
DEM	Digital Elevation Model
SOCA-CFAR	Smallest of Cell Averaging-CFAR

CHAPTER 1

Introduction

1.1. Motivation

With the increasing relevance of climate change new concepts for mobility are required. A change in mobility demands not only decreasing carbon emissions but also increasing flexibility in order to remain competitive in an international context. With the strategic plan "Transport 2050" the European Commission sets targets to increase mobility in the European Union and to reduce emissions at the same time. Among these targets is the requirement to shift freight transport from road to rail and water. This is to be implemented by 2050 for 30% of freight transports over 300km [53]. Within the European Union, Germany proposed a national master plan specifically for the future of inland navigation in which the importance of inland navigation for a sustainable future is further highlighted. One important aspect is to increase the transport of goods by inland navigation to reach 12,3% of total transported goods until 2030 [39]. These requirements can be set against a background of increasing congestion on the roads and at the same time overcapacity of transport by water. In the context of these changes, waterborne transport, and in particular inland navigation, must prove to be competitive both economically and in terms of sustainability [51].

The automatisation of inland navigation vessels up to the possibility of an unmanned operated or autonomous vessels can play a vital role to full fill this requirements. There are expected to be huge advantages in operating USV in comparison to traditional operated vehicles. With the automatisation of the vehicles comes an reduction of crew members to control a vehicle and hence the operational cost are reduced. Furthermore the reliability and flexibility in harsh or otherwise sophisticated environments increases. Particularly in shallow waters this flexibility allows

an adaption of the USV towards the requirements of this environment [37].

The technological innovation efforts of the inland navigation sector comprised mainly the development of low-emission engines [51]. However, research into the field of USV has been pursued over the past two decades. Even so, most current research is conducted on relatively small vehicles with limited capabilities regarding autonomy, endurance, payload and power outputs and experimental research remains an important topic for the future [37].

Therefore, designing a vessel specifically for autonomous operation to evaluate technologies for the next generation of inland navigation vessels is an important step towards further innovations in this sector. More specifically, the vessel should be able to sense its environment and operational state to enable situational awareness and autonomous decision making, possibly with the application of machine learning. Simulated environments provide an opportunity to facilitate the process of sensor data generation by means of virtual sensors. Therefore a software architecture that supports both real world and virtual appliance is of great importance in the design process of such a vessel.

1.2. Previous work

Within the research project Guiding and Assistance Systems to Improve Safety of Navigation on Inland Waterways (LAESSI), a driver assistance system was designed and implemented on an inland navigation vessel with the focus on a bridge collision warning system and instruments to visualize the internal and external state of the vessel. To localize the vessels position, a land based corrected Global Navigation Satellite System (GNSS) signal was used together with electronic charts and further information transmitted from a land based station as for example temporarily restrictions caused by construction sides and the current water level. Internal sensors comprise a inertial measurement unit, a height sensor and turn indicator. The sensors data together with the GNSS data are processed and checked for integrity as errors in the positing data directly influences the rudder commands in the chosen system architecture [45]. The sensors in this research project however do not allow the vessel a land station independent representation of its environment and thus the vessel depends on an established connection to a land based station to function properly.

A first step towards a higher degree of automatisation with the focus on reducing crew members to operate a vessel was laid by the KU Leuven with the vessel "Cogge". A design for an automated vessel is proposed with focus on the three key technological design aspects industrial relevance, vessel-environment interactions and the motions control of the vessel. The design is in accordance with an EU co financed vessel design project Watertruck+ [56], in which standardised and

modular small barges and push boats allow a high degree of flexibility especially in small waterways. The sensors of the "Cogge" comprising GNSS, IMU, LIDAR and stereo image sensors. The software for the motion control is organized in a multi-layered software design. To determine the route, waypoints are generated based on an open source map which then can be utilized by the middle level control. Software for the middle level control is provided by the Mission Oriented Operating Suite-Interval Programming (MOOS-IvP) system [4], a set of open source software modules specifically for the operation of autonomous marine vehicles. To reduce the error between the current and the desired speed and direction, a Proportional Integral Derivative (PID) controller compares the GNSS and IMU sensor values with given data of the MOOS-IvP. The PID controller connected with a downstream Programmable Logic Control (PLC) is implemented in the lower level control. Experiments conducted with this platform however has tended to focus on following generated waypoints with optimizing course deviations[41]. The integration of sensors beyond IMU and GNSS and the ability to manoeuvre in confined spaces as necessary in weir and lock systems remains open. Furthermore, the outlined design is lacking a RADAR system to generate spatial data for objects further away than the range of the LIDAR. That also leaves open the question how to merge the RADAR and LIDAR data at its transition.

With increasing interest in autonomous driving especially in the field of automotive during the recent years, the research into automotive systems that integrates a wide variety of sensors is considerably more detailed compared to the inland navigation sector or more generally compared to water bound vehicles. A software-stack framework that is called "Autoware" provides an example of such a system designed to enable autonomous driving of vehicles. It is based on Robot Operating System (ROS), an opensource middleware framework developed for robot applications and is open source itself. The required sensors that are used by the software can be purchased on the market and comprise a LIDAR, a camera, GNSS with Real Time Kinematics (RTK) in the proposed system. On the software side, the tasks of algorithms are divided into the classes scene recognition, path planning and vehicle control. For each of these classes a set of algorithms is proposed. The software is run on an Nvidia Graphics Processing Unit (GPU), which allows for example for a faster object detection compared to Central Processing Unit (CPU) based computation [26]. In a subsequent paper the migration of the software stack to an embedded system on board of a vehicle is proved and insofar its hardware flexibility is highlighted [27]. The integration into an inland navigation vessel with its requirements and external influences remains open.

1.3. Outline

The present document is organized in five chapters. The first chapter (1) gives an introduction to the context of this thesis as well as a brief overview of previous

research within the field of software architectures for inland navigation vessels.

The second chapter (2) explains LIDAR and RADAR systems as well as the algorithms that are used to process its data. For the LIDAR, direct, cluster based, hierarchic and density based methods are elaborated on that can be applied to cluster pointclouds and derive a structured representation of the environment in the around the sensor. To process the data of the RADAR, algorithms for a fixed threshold and dynamic threshold signal filtering are presented. Three different implementations of the dynamic threshold are further outlined.

Oriented on the V-Model for the development of safety related software, chapter three (3) elaborates on the design of the software architecture for virtual sensors. First the overall system is designed with respect to appliances beyond this thesis. This is followed by a detailed description of the part that is implemented inside the system with a focus on the four sensor types that are integrated into the system.

The system as designed in chapter three (3) is tested and discussed in chapter four (4). Module and integration tests are conducted with emphasis on the performance of the system with the latency as an criteria. With a subsequent software validation, the implemented system is tested against the requirements.

With a summary and outlook given in chapter five (5), this thesis is concluded.

CHAPTER 2

Fundamentals

This chapter aims to give an introduction to an GNC system as used commonly for USV and according sensor processing algorithms to represent the GNC systems environment. In section 2.1 the tasks of the three subsystems the GNC consists of are briefly explained. Hereafter, section 2.2 explains the notation for rotations as used in Unity3d. Section 2.3 outlines an algorithm that is used to extract lines of an image. The section is followed by an explanation of the fundamentals of distance measurement with LIDAR in section 2.4 and object detection with RADAR in section 2.5.

2.1. The USV and Guidance, Navigation and Control

Every USV includes six basic elements which are the hull and auxiliary structural elements, propulsion and power systems, GNC systems, communication systems, data collection equipment which are the sensors of the USV and the ground station. According to various fields of applications, elements can be added. To operate an USV autonomously, GNC systems as depicted in figure 1 are of outstanding importance. The GNC subsystem is divided into three main parts, namely in systems that full fill the purpose of navigation, guidance and control objectives. As these subsystems are highly interlinked with each other, a malfunction of one of them can compromise the performance of the whole USV.

Connected to the onboard sensors, the navigation subsystem identifies future states of the USV and represents the USV's current state as well as the environment. Sensors used to perceive the environment includes passive visual and infrared sensor as well as active perceptions methods such as LIDAR, RADAR and Sound Navigation and Ranging (SONAR). These sensors can also support the process of state estimation of the USV as the main reference for localisation provided by the

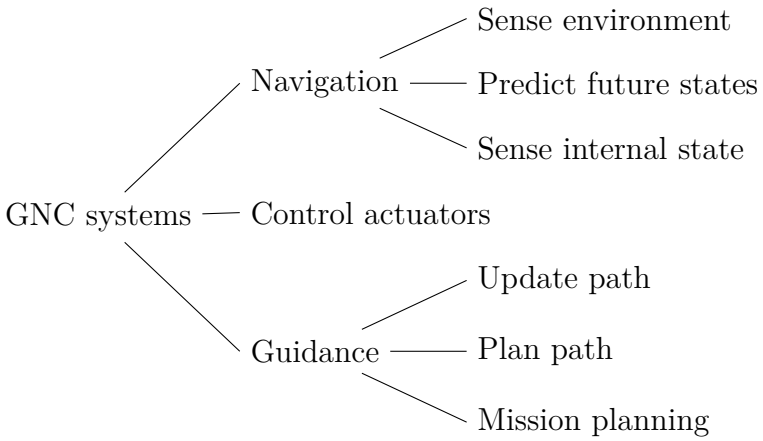


Figure 1.: GNC of USV [37].

GNSS and IMU may be disturbed by the environmental influences. As clutter and disturbances like waves and wind are practical problems and lead to undetected targets in the application of sensors, these effects have to be filtered by appropriate methods to avoid the provision of misleading information.

With the processed data of the navigation subsystem, a map with obstacles is to be generated as a first step in a so called mission planning [37]. This obstacle map then allows for different methods the application of path finding algorithms to provide a trajectory from the actual position to a desired position. Two main distinctions between path planning algorithms can be made by dividing them into global and local approaches. Due to the problems of considering unexpected or dynamic obstacles when calculating a path, global approaches are more suitable for static environments or offline calculations. Further limitation in the application of these algorithms can be found in the considerably high computational resources and time to find a path. Local path finding algorithms provide the possibility of taking dynamic objects into account as they use updated sensor data and its computational effort allows real time behaviour [5]. An algorithm applying so called potential fields can be assigned to a local path finding approach. The object moves in a field of forces that originate from the kinematics of the object itself, repulsing forces of the obstacles and attracting forces from the spatial point that is to be reached. By finding a path that maximizes the potential, the object is supposed to reach the required spatial point. However, it has to be noted, that local path finding approaches are prone to find only local minima of optimal paths because they do not take into account the whole scene [29].

With the information of both the guidance and the navigation subsystem the control subsystem determines the control forces and moments to be generated at the actuators in accordance with the desired control objectives and the USV capabilities [37]. The mathematical model to describe the ships characteristics can be sufficiently described by a three degrees of freedom model especially for inland navigation, as the ships rarely encounter large waves [36]. Therefore, the influence of heave, pitch

and roll can be neglected and only the surge, sway and yaw is described by the equations of the model. Since most ships are equipped with a rudder and one or more propellers or water jets, only surge forces and the yaw momentum can be manipulated by the control system to obtain the control objectives, thus the ship model can be called under actuated. With equation 2.1.1 derived in [11], the surge and yaw displacement as well as the yaw angle are described in a earth fixed frame by x, y and ψ whereas equation 2.1.2 describes the surge, sway and yaw velocities by u, v and r.

$$\dot{x} = u\cos\psi - v\sin\psi, \quad \dot{y} = u\sin\psi + v\cos\psi, \quad \dot{\psi} = r \tag{2.1.1}$$

$$\begin{aligned}
\dot{u} &= \frac{m_{22}}{m_{11}}vr - \frac{d_u}{m_{11}}u - \sum_{i=2}^{3}\frac{d_{ui}}{m_{11}}|u|^{i-1}u + \frac{1}{m_{11}}\tau_u + \frac{1}{m_{11}}\tau_{wu}(t),\\
\dot{v} &= \frac{m_{11}}{m_{22}}ur - \frac{d_v}{m_{22}}v - \sum_{i=2}^{3}\frac{d_{vi}}{m_{22}}|v|^{i-1}v + \frac{1}{m_{22}}\tau_{wu}(t),\\
\dot{r} &= \frac{m_{11}-m_{22}}{m_{33}}uv - \frac{d_r}{m_{33}}r - \sum_{i=2}^{3}\frac{d_{ri}}{m_{11}}|r|^{i-1}r + \frac{1}{m_{33}}\tau_r(t) + \frac{1}{m_{33}}\tau_{wr}(t)
\end{aligned} \tag{2.1.2}$$

The ships inertia as well as the added masses are denoted with m_{xx} and with d_{xx} the hydrodynamic damping in surge, sway and yaw is represented. Environmental disturbances are modelled with $\tau_{wu}(t)$, $\tau_{wv}(t)$ and $\tau_{wr}(t)$, caused by the influence of waves, wind and current. Higher non-linear terms are neglected to keep the model as compact as possible. The available control variables are the surge force τ_u and the yaw moment τ_r. As depicted in figure 2, the actual position of the ship is assumed to be specified by a point $M[x, y]$ and a direction vector u which is related to the ships spatial orientation. The desired position is defined by a point $M_d[x_d, y_d]$ and a related vector u_0. The control objectives can be formulated by equation 2.1.3.

$$\begin{aligned}
&x_e = x_d - x, \quad y_e = y_d - y, \quad \psi_e = \psi - \psi_d\\
&z_e = \sqrt{x_e^2 + y_e^2}, \quad \psi_d = arcsin\frac{y_e}{z_e}
\end{aligned} \tag{2.1.3}$$

The distance difference from M to M_d are expressed with x_e and y_e which results in the summed distance z_e. Hereafter, the angle deviation can be formulated by ψ_d. Therefore, it becomes clear that in order to obtain a controllable system, the ships navigation subsystem has to provide the position and heading of the ship, the environmental data such as wind speed, the current and the waves direction as well as the scale [59]. The desired state M_d has to be calculated by the guidance subsystem. At least, after the computation of a suitable control, the actuators that generate the surge force and yaw moment are to be controlled accordingly [11].

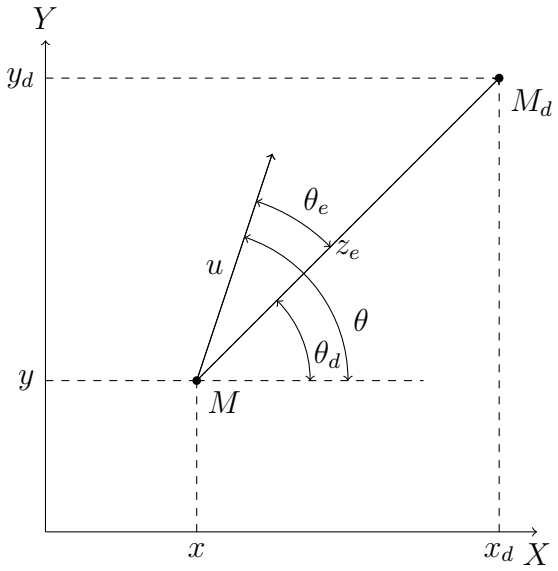


Figure 2.: Approximation of a ships position M towards a desired position M_d by [11].

2.2. Quaternions

To describe fluid rotations of rigid bodies in a 3D space, and subsequently visualize the results in a simulation tool, a suitable mathematical representation of the rotation has to be found. Accepted as the most intuitive way are the so called Euler Angles. Rotations are described with the rotation matrix $\boldsymbol{R}_{()\boldsymbol{\psi},\boldsymbol{\theta},\boldsymbol{\phi})}$as shown in equation 2.2.1.

$$\boldsymbol{R}_{(\boldsymbol{\psi},\boldsymbol{\theta},\boldsymbol{\phi})} = Rot[z,\psi]Rot[x,\theta]Rot[z,\phi] \tag{2.2.1}$$

Matrix $\boldsymbol{R}_{(\boldsymbol{\psi},\boldsymbol{\theta},\boldsymbol{\phi})}$ is composed of the three rotation ψ, θ, ϕ around the three carthesian axes x, y, z. However, calculating this rotation matrix and the respective rotated vector for every incremental step in order to reach a fluid rotation until a required degree is obtained is very demanding in terms of computational power. Additionally, in case one of the rotation degrees ψ, θ, ϕ are zero, one degree of freedom is lost as two rotations about the same axis take place. This phenomenon is called gimbal lock. As a third back draw, the order in which the rotation about the axis are executed are important to the final result. Thus, a more suitable notation for rotations can be found in the so called quaternions. Quaternions follow a specified quaternion algebra. For instance operations on it are not commutative. Any quaternion can be written as depicted in equation 2.2.2.

$$\begin{aligned} q &= d + ai + bj + ck \\ q &= cos\theta + sin\theta(s_x i + s_y j + s_z k) \end{aligned} \tag{2.2.2}$$

The three dimensional complex tuple *i, j, k* now forms a hyperspace that is in

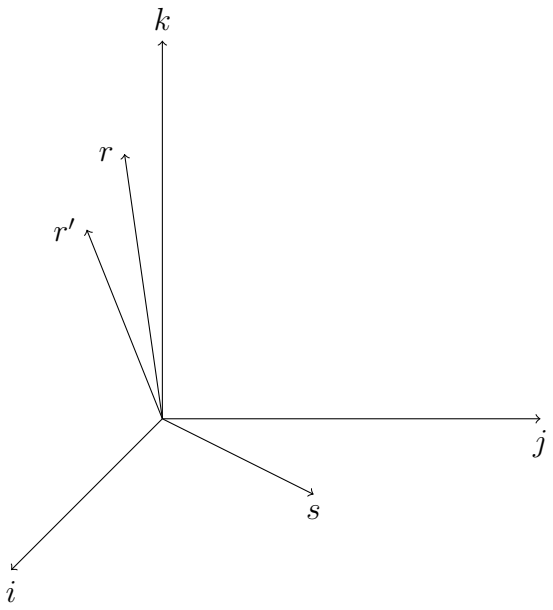


Figure 3.: A rotated vector in a Quaternion 3D hyperplane after [9].

the four dimensional space that exists with the real number d. a, b, c or s_x, s_y, s_z respectively define the rotation axis around which is rotated and represent a vector in the hyperspace as shown in figure 3. The rotation itself is defined by the angle θ. By specifying a vector r as shown in equation 2.2.3, the rotated vector r' can be calculated with two multiplications with the normalized quaternion as in equation 2.2.4 [9].

$$r = r_x i + r_y j + r_z k \tag{2.2.3}$$

$$r' = qrq^{-1} \tag{2.2.4}$$

2.3. Line Extraction with the Hough Transform

One algorithm that is widely used over time to enable a computer the extraction of patterns in images is the Hough transform. Let a greyscale image consist of points that are defined in a two-dimensional cartesian coordinates system with some of the points fall on a line. Now given a quantity of points X, Y on one line, equation 2.3.1 is fulfilled for the slope m and intercept b of that line.

$$f((X, Y), (m, b)) = Y - mX - b = 0 \tag{2.3.1}$$

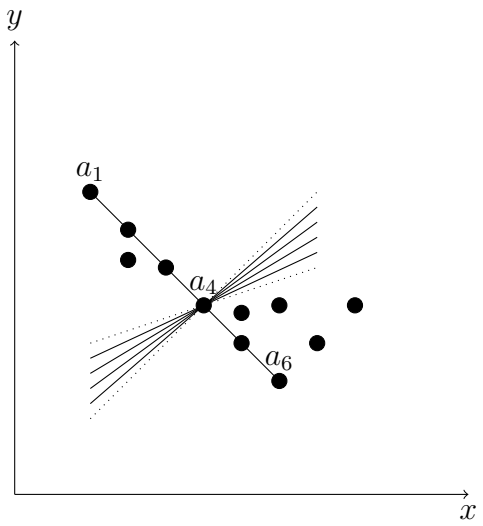


Figure 4.: Points in image space

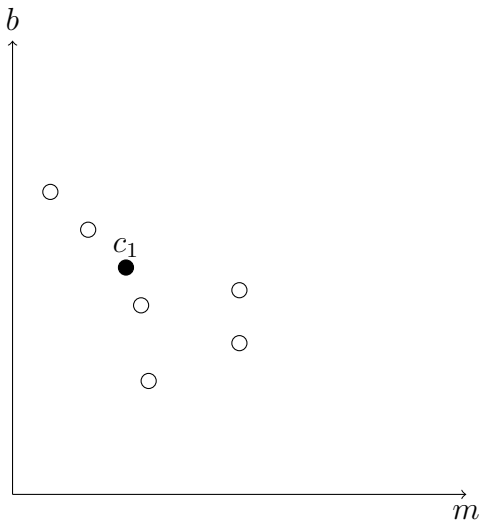


Figure 5.: Lines in feature space

Taking a single point x, y separately, the lines described by the quantity of M and the intercept B through that point full fill equation 2.3.2.

$$f((x, y), (M, B)) = y - Mx - B = 0 \tag{2.3.2}$$

The equation describes every possible line through the point x, y. Computing the quantity M, B for several points on one line, all sets m, b of values are different from each other apart from the one that describes the line on which the points lie. Because this value occurs more often, the line can be detected. The Hough transform and the line detection is visualized in figure 4 and 5. With the points a_1, a_4 and a_6 in figure 4 a line is described in the Hough image space. The image space is a picture that has been preprocessed with a edge detection with for example a Canny edge detection. For every point in the image space, the quantity M and B is build so that it suffices equation 2.3.2. For point a_4 a few of the possible lines are shown. Every found combination of m and b is written to the Hough feature space. In the feature space, the intercept of lines is plotted against the slope. In the feature space, the line that is formed by a_1, a_4 and a_6 in the image space is plotted in black to symbolize that the corresponding slope and intercept occurs more often. Because in the feature space the values are collected that define the lines in the image space, it is called an accumulator. The more often one set of values is found, the more likely it is that a line exists in the image space. More precisely in the feature space, the slope and intercept is plotted in a polar coordinate system which is omitted here for reasons of simplicity. [23]

2.4. Distance Measurement with LIDAR

With the rapid development of lasers for optical distance measurement, today the LIDAR technology is widely used in different fields of applications. After an introduction to the working principle of a LIDAR, this section continuous by detailing four clustering algorithms that can be applied on the LIDAR data. In subsection 2.4.1 the Hough algorithm for the three-dimensional space is explained followed by the partition based K-Means algorithm in subsection 2.4.2. Hereafter, subsections 2.4.3 and 2.4.4 explain the hierarchic Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) and density based Density Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm respectively.

A classification based on the transmission waveform and data processing method divides the LIDAR technology into pulsed, continuous wave, pulse compression, moving target display, pulse Doppler and imaging LIDAR. All of these varieties are based on the fundamental principle of a Light Amplification by Stimulated Emission of Radiation (LASER). The wavelength of the laser is in the range of ultraviolet, infrared or visible light. Typically, a wavelength of about 1000nm or less is used. One advantage that a pulsed laser has over a continuous laser is the higher light intensity for a period of time. A typical measurement cycle according to the pulsed laser method comprises three integrations. A laser pulse is emitted and the received radiation is registered via integration one. After the time t_{pulse} is elapsed, in the first integration the backlight and the first part of the reflected pulse is saved. Hereafter, integration two sums up the backlight and the last part of the reflected pulse. Integration three solely measures the backlight over a certain period of time. With equation 2.4.1 derived in [48] and in [12], the quotient Q_{ToF} can be calculated. The backlight or noise is subtracted by integration three from the reflected light measured in integration one and two. With the pulse length t_{pulse}, t_{ToF} can be calculated by (2.4.2). By multiplication with the speed of light c, the now known value t_{ToF} results in the distance of the object from which the light was reflected.

$$Q_{ToF} = \frac{integration2 - integration3}{integration1 + integration2 - 2 * integration3} \tag{2.4.1}$$

$$t_{ToF} = t_{pulse} * Q_{ToF} \tag{2.4.2}$$

$$d = \frac{c}{2} * t_{ToF} \tag{2.4.3}$$

To obtain a point cloud and scan the environment of the LIDAR, the emitted light has to be directed by some means. Different scanning methods exist of which one is a motor-driven rotation of a mirror, which reflects the laser and hereby allows sampling of the surrounding of up to 360°. As moving parts are part of this method, friction and therefore a reduced durability are inherent to this approach. However, because of the technical simplicity this implementation is widely used. A typical

mechanical LIDAR scanner with the aforementioned laser as a source, comprises the a servo motor as an actuator and a mirror as the rotational element.

The segmentation of the LIDAR data takes a vital role to extract the necessary information used for the subsequent object classification. A number of segmentation algorithms were proposed in the last years which can be divided into direct and indirect approaches. Given a point cloud with artefacts shaped in regular geometric shapes, an example for a direct approach is given by the Hough transformation. Geometric parameters can be extracted directly from the point cloud data besides the segmentation process. Indirect segmentation methods apply for example progressive algorithms (over time improving approximations of a complete solutions with intermediate results [1]) on the point cloud data to compute spatial proximity and geometric derived values as for example local surface normal vectors. Common algorithms for an indirect data segmentation by clustering without feature extraction comprise methods based on partitioning, hierarchy and density.

2.4.1. Hough Transform Algorithm (direct approach)

The Hough Transform Algorithm for the 3D space works similar to the Hough Transform Algorithm in the 2D space which is frequently used in image feature extraction. The 2D Hough algorithm is explained in section 2.3. Let a planar feature in the 3D carthesian space be described by a function F as shown in equation 2.4.4, it can be converted into angle information to the form in equation 2.4.5.

$$F = ax + by + cz + d = 0 \tag{2.4.4}$$

$$\begin{aligned} &\rho = x\cos\theta\sin\phi + y\sin\theta\sin\phi + z\cos\phi \\ &\theta \in [0°, 360°], \phi \in [-90°, 90°] \end{aligned} \tag{2.4.5}$$

The normal vector n of the planar feature is described by the parameters θ, ϕ and ρ as depicted in figure 6. Accordingly, the Hough space consist of these parameters. The algorithm (accumulator) now iterates through the point cloud data and maps the possible normal vectors of each point in its feature space (Hough space). Hence, every time a point is on the planar feature, the density of the parameters (θ, ϕ, ρ) in the feature space related to n will increase. Points with a high density in the feature plane therefore represent a plane in the 3D carthesian space and the feature is extracted. However, for each point in data, there are $x_\theta * x_\phi$ possibilities to be mapped in the feature space. With y points in the point cloud the accumulator needs $y * x_\theta * x_\phi$ iterations to finish the process. For large databases, the process therefore requires many iterations and the computational power has to be sufficient to cluster the database in the required time frame of the application.

2.4.2. K-Means algorithm (partition based)

The K- means partitioning algorithm follows a very straight forward method and is divided into three steps. The first step is to find a center point for each cluster, which

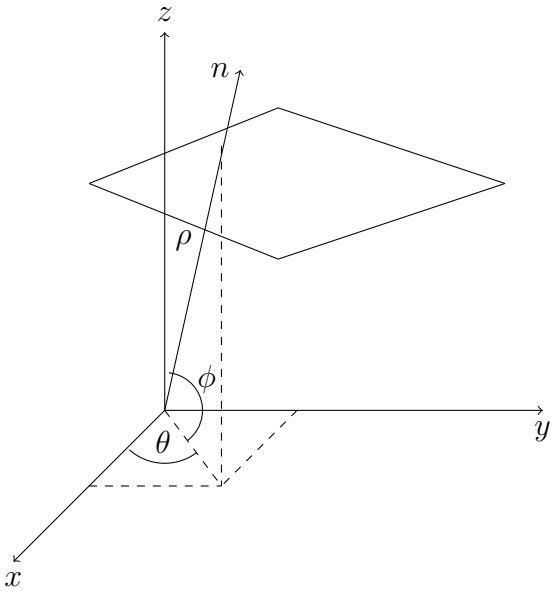


Figure 6.: A planar feature described in the 3D hough image space [5].

can be done randomly. The distance from each point to the center point of each cluster is calculated in a second step. In a third step, the average coordinates from each point of a cluster is calculated and defined as the new center of the according cluster. Step two and three can be iterated until a difference criterion between the former and actual cluster center is found. Therefore to reach a certain accuracy, many iterations can be necessary and the computational effort for large databases is high.

2.4.3. BIRCH Algorithm (hierarchic)

The BIRCH algorithm is used for very large datasets which are to be processed in a limited time with very small amount of used memory space. In most cases, one single scan of the data which is to be clustered is sufficient. Furthermore, noise is handled effectively [57]. Based on the Clustering Feature (CF) and the hierarchical Clustering Feature Tree (CFT), the CF is defined by the triplet $(N_i, \vec{LS}, \vec{SS})$, in which N_i represents the i data points in the cluster, $\vec{LS}$ the sum $\sum N_i$ and $\vec{SS}$ the sum of squares $\sum N_i^2$. The cluster features can be added to each other which is expressed by equation 2.4.6.

$$CF1 + CF2 = (N1 + N2, \vec{LS}1 + \vec{LS}2, \vec{SS}1 + \vec{SS}2) \tag{2.4.6}$$

The CF are stored in a CFT with Root, Nonleaf, and Leafnotes and the MinCluster as depicted in figure 7. B describes the maximum branches of the Nonleafes and L the maximum branches of the MinCluster. The parameter T limits the radius of the cluster. The algorithm consists of five consecutive steps. First, the nearest CF

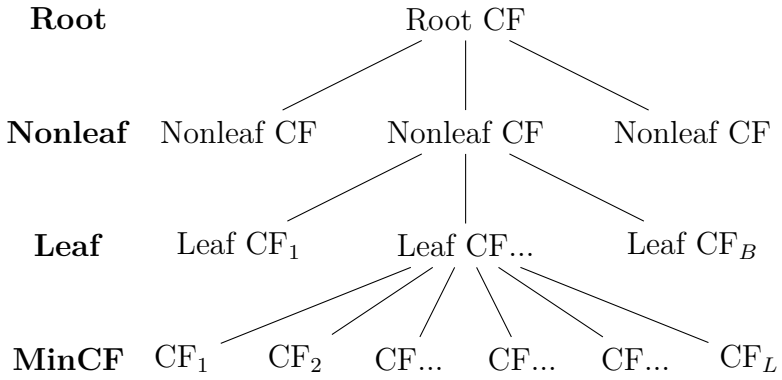


Figure 7.: A CFT derived from the BIRCH Algorithm [5].

node to the new sample is found. If, second, the distance between the new sample and the CF node is less than the threshold T, insert the sample. Otherwise, and if the CF branches (MinCluster) of the leaf node are below the threshold L, insert a new MinCluster. If the threshold requirement L is not met, in a fourth step divide the leaf into two new leafs. The two MinCluster furthest away from each other are selected for the new leaf nodes. The new sample and the already given samples have to be assigned to the according leaf hereafter.

The main advantage is that no iteration over the datapoints is necessary, as each datapoint is read in only once. As the CFT only points to the data that is saved, even high amounts of data can be processed. However, due the limited CF per node, the algorithm is prone to over segmentation and is also called a micro clustering approach. Therefore, created cluster may not represent the real world situation especially for large target objects in the measured data [5]. Additionally due to distance restriction of the CF, the shape of the cluster data tends to be spherical for large target objects.

2.4.4. DBSCAN (density based)

With the DBSCAN algorithm a method is proposed with which arbitrary shaped targets can be clustered even in an environment with present noise. It is a density based approach and efficient for large spatial databases. The algorithm takes advantage of the characteristic of a shape or cluster, that the points of the cluster provide a higher density than the noise outside of a cluster. Therefore each point inside a cluster requires to have a minimum of points $MinPts$ inside its neighbourhood Eps. The shape of a neighbourhood is to be determined by the function of distance of two points with $dist(p, q)$, where Eps is the maximum distance between two points p, q to consider them as neighbours. For example for an Euclidean distance the shape of a neighbourhood would be spherical and for a Manhattan distance, the shape would be rectangular. Equation 2.4.7 describes this relation with N_{Eps} describing the Eps neighbourhood and D the dataset of points in a k-dimensional space to be

considered.

$$N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\} \tag{2.4.7}$$

A point p that full fills the requirement of $MinPts$ points q in its neighbourhood is considered to be directly-density-reachable with the formal definition described by equation 2.4.8.

$$p \in N_{EPS}(q) \wedge |N_{Eos}(q)| \geq MinPts \tag{2.4.8}$$

However, because a data point p at the border of a cluster may not full fill the requirement of $MinPts$ points inside its neighbourhood but is nevertheless inside the cluster, its relation to a point q is defined by the expression density-reachable. A point p is density-reachable, if there is a chain of points $p_1, ..., p_i = q, p_n = p$ with p_{i+1} directly-density-reachable from p_i. A third definition is required to describe the relation of two points p and q both at the border of a cluster. They are density-connected if a point $o \in N_{Eps}$ is density-reachable from p and q [14] [47]. Figure 8 depicts the aforementioned relations with the points B, A, C and N. N can be considered as noise and is unrelated to a cluster because it is not connected to another points by any given definition. The data points A are directly-density-reachable from each of the red points. The data points B and C are density-reachable from A and the other red points are density-connected to each other by the red points. The found cluster now consists of the red and yellow points.

As density based algorithms as the DBSCAN depend essentially on the closeness Eps of the data points to each other, spatially distributed cluster and varying density results in large inaccuracies [7]. Therefore, in an environment with these characteristics an adaptive value for Eps is required. A linear threshold expressed with equation 2.4.9 has been experimentally proven to be feasible. The arc length L is calculated with the angle resolution θ and the distance d_i and results in Eps together with the precision factor n. Additionally, because not only the distance of points to each other, but the density of points inside a cluster can be reduced by increasing distance, the minimum of required points inside a cluster may be adapted with a linear equation related to the distance [21].

$$\begin{aligned} L &= \frac{\theta \pi d_i}{180} \\ Eps &= nL \end{aligned} \tag{2.4.9}$$

2.5. Object Detection with RADAR

After an introduction to the fundamental formulas that describe RADAR systems, this section continuous with four different approaches to filter the RADAR data. in subsection 2.5.1, a constant threshold is explained to separate noise from the signal. Continued by subsection 2.5.2, a dynamic threshold calculation is introduced which is the foundation for subsection 2.5.3 and 2.5.4. In this subsections, derivatives to the afore mentioned dynamic threshold calculation are explained.

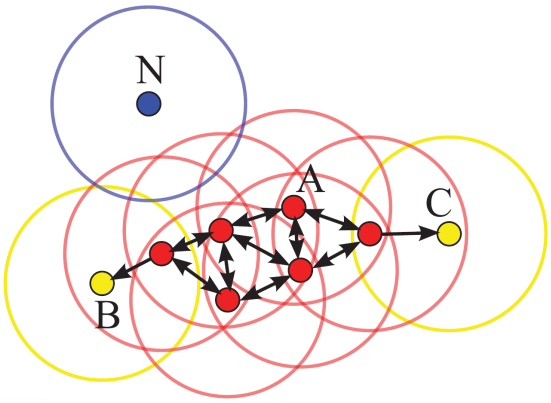


Figure 8.: Clustered data with DBSCAN and noise [47].

A millimetre wave RADAR can estimate distances by means of electromagnetic waves with a frequency between 30GHz and 300GHz and wavelength between 1-10mm. An electromagnetic wave is emitted and reaches a possible target with the energy S_e (equation 2.5.1). In equation 2.5.1, g represents the antenna gain, P_S the power of emittance in Watts and r the distance from the antenna to the target.

$$S_e = \frac{gP_s}{4\pi r^2} \qquad [\frac{W}{m^2}] \tag{2.5.1}$$

The electromagnetic wave is then reflected with the energy S_s according to equation 2.5.2 with r similar to 2.5.1 and A_e being the RADAR cross section calculated by 2.5.3 [42].

$$S_s = \frac{A_e S_e}{4\pi r^2} \qquad [\frac{W}{m^2}] \tag{2.5.2}$$

Equation 2.5.3 can be described as the ratio of the target surface A_{pl} to the wavelength γ of the emitted wavelength. Therefore, the larger the wavelength and the smaller the surface area of the target object, the smaller the reflected energy and the higher the signal to noise ration. For practical considerations, in most cases values with sufficient accuracy can be calculated by approximating the target area as planar[42]. For further specification, the power of the radiation depends on the size, shape and material of the target [5].

$$A_e = 4\pi \frac{A_{pl}^2}{\gamma^2} \qquad [m] \tag{2.5.3}$$

Two operational principles can be distinguished of which the first is the continuous wave system RADAR and the second the pulse system RADAR. In the continuous wave system the reflectance of the target object is received during the emittance of waves. In the pulsed system, a wave is emitted intermittently and during the emittance breaks, the reflectance is received. Inherent to both systems is the importance

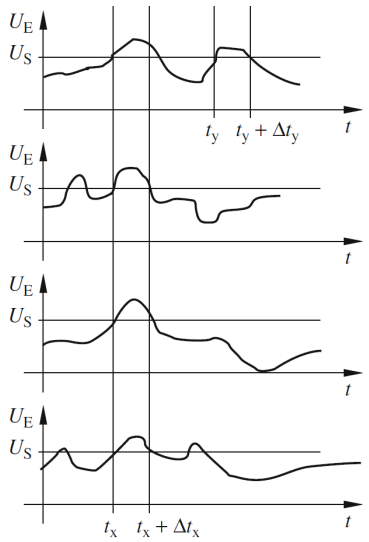


Figure 9.: Four datasets to distinguish the target from noise and clutter [42].

of signal processing to increase the target detection in a non uniform environment and improve target position estimation.

2.5.1. Constant Threshold

The most straight forward approach is to empirically determine a threshold to separate the objects reflectance signal from the noise and clutter of the measurement. The higher the threshold, the lower the probability of false alarms becomes. However, with a high threshold, the probability to ignore targets becomes higher, too. Therefore, the level of the threshold is vital in this approach. To add further robustness, several measurements of the same object can be compared and only data points which are consistently above the threshold can be specified as an object representation. Hence, the more measurements are made, the higher is the certainty with which objects can be determined. In figure 9, four different measurements of the same scene are depicted. On the first axis the time t is plotted and the second axis represents the reflected signal strength U_E. Only the corresponding value of t_x to $t_x + \Delta t_x$ is above the threshold U_S in all measurements. Thus, a target reflectance can be associated to this period of time with some certainty. In contrast, the time t_y to $t_y + \Delta t_y$ exceeds the threshold only once and can be classified as noise or clutter [42]. Due to the fixed threshold, environments with changing noise intensity results in varying probabilities for false alarms. Additionally, the targets reflectance may change due to positional deviations and resulting surface area changes and thus further raise the false alarm probability.

2.5.2. CFAR

By taking the Signal Noise Ratio (SNR) as a threshold rather than the absolute value of the signal, an adaptive signal threshold can be realized and the probability for an false alarm is reduced even in situations with a high noise signal strength. First, the noise of the measurement has to be determined and normalized. Specifically in naval applications, the noise can be approximated with different Probability Density Functions (PDF) of which one is the Rayleigh distribution [25]. The PDF of the Rayleigh distribution is shown in equation 2.5.4 with the noise x and b proportional to the mean of the noise μ according to 2.5.5.

$$P(x) = \frac{x}{b}^{-\frac{x^2}{2+b^2}} \tag{2.5.4}$$

$$b = \sqrt{\frac{2}{\pi}} + \mu \tag{2.5.5}$$

To express equation 2.5.4 independent of the noise level, b can be substituted by $y = x/b$ which results in equation 2.5.6.

$$P(y) = y^{-\frac{y^2}{2}} \tag{2.5.6}$$

The probability for a value y is now only dependent on the SNR of the target signal to the mean noise of the environment. A system diagram of the necessary operations is provided by figure 10. A mean value μ of the noise is calculated with one or more sampling pulses and compared with the input signal x. It is not required to take additional sampling pulses to the already taken samples, because the noise can be extracted from the present measurements. This can be either done by dividing the area of interest into cells and measure the noise in cells surrounding the target by assuming that these cells only consist of noise [17]. Another approach is to use the measured signal levels before and after the target reflectance. Therefore, it has to be assumed that the area before and after the target represents solely noise [5]. After the noise extraction, the mean value μ of the noise is estimated and compared to the input signal x. The hereby resulting value y is given to the detector, which compares the value to the SNR threshold U_0. Either the signal is above the determined threshold and the output results in a target signal (one) or the threshold is not exceeded and therefore no target signal is detected (zero). The probability for a false alarm therefore can be set through the threshold.

2.5.3. OS-CFAR

Considering the possibility of different noise and clutter levels in the sample, the aforementioned method with a very basic noise calculation approach can reach its limits in real world applications. The method as proposed in [46] resembles the ideas of image processing. More specific, the clutter and noise power estimation

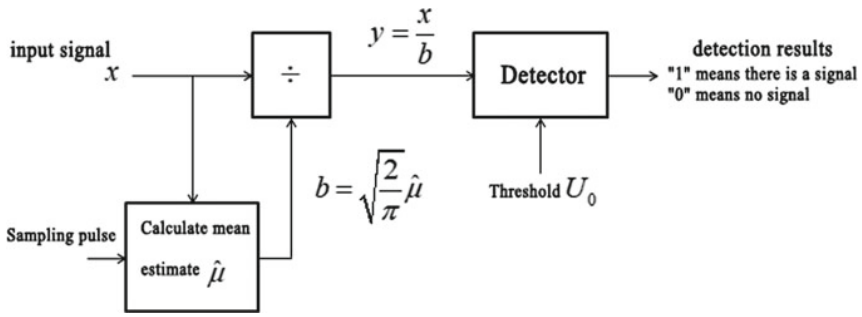


Figure 10.: Single target CFAR with fixed threshold [5].

that for example derives one value by averaging the whole noise sample is replaced by an arithmetic averaging procedure that is applied already in image processing applications. Especially for dense target situations, the proposed method provides advanced target extraction capabilities. The most significant change to common CFAR approaches is the application of order statistics to derive a noise value close to the field of interest. The field of interest is divided into cells of which the centre cell is called Cell Under Test (CUT) as depicted in figure 11. The aim is the decision whether there is a target present in the cell or not by comparing the signal strength of the cell with its surrounding cells. The green cells right and left of the CUT are called guard cells and are lined up in a one dimensional space in this example. In a three dimensional application they would build a sphere around the CUT accordingly. They are ignored in the measurement, because the signal energy can affect adjacent cells and may effect the noise calculation. The signal strength of the cells $x_1 ... x_N$ and $y_1 ... y_M$ are most important for the noise calculation and are processed by the order statistic process in such a way that $x_{(1)} \leq ... \leq x_{(k)} \leq ... \leq x_{(N)}$ and $y_{(1)} \leq ... \leq y_{(k)} \leq ... \leq y_{(N)}$. Hereafter, these values can be described by a PDF of order statistics. One value $x_k, \quad k \in \{1, 2, ..., N\}$ or $y_k, \quad k \in \{1, 2, ..., M\}$ is then selected and used as an estimation for the average noise power Z as shown in equation 2.5.7.

$$Z = X_{(k)} \tag{2.5.7}$$

By scaling Z with the factor T, an adaptive threshold S for the comparator is found (equation 2.5.8).

$$S = TZ \tag{2.5.8}$$

In figure 11, α resembles the scaling factor T of equation 2.5.8 and with β the threshold can be further adjusted. A comparator thereafter compares the value of the CUT to the threshold and the decision is made, whether there is a target (one) or only noise (zero) in the according cell. The algorithm then iterates through the cells until each cell is set to zero or one. The advantage of multi target detection in this approach resulting from the independency of the mean value of the clutter comes by the disadvantage of high computational effort to calculate the threshold. The probability that a random noise variable Y_0 of the CUT is interpreted as an echo of the target can be expressed with equation 2.5.9. To obtain a required false

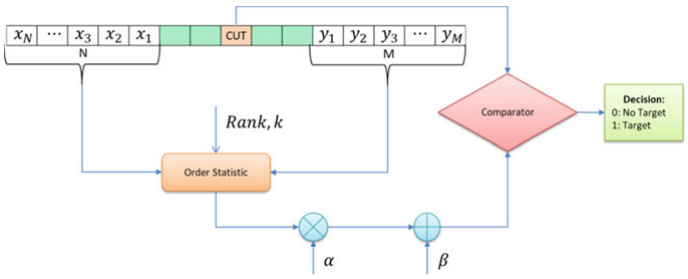


Figure 11.: Multi target OS-CFAR, threshold calculated by ordered statistics [17].

alarm rate, this equation has to be solved to derive the scaling factor T.

$$P_{fa} = P[Y_0 \geq TZ] \tag{2.5.9}$$

As aforementioned, in sea borne applications, the Rayleigh function can be used as an estimation to clutter and therefore describes the PDF of Y_0 sufficiently. However to proof the independency of the mean value of all noise values in the threshold calculation, a simple exponential distribution for the noise following $1/\mu e^{-x/\mu}$ is sufficient. The PDF of $Z = x_k$ is shown by equation 2.5.10.

$$P_{x_k} = \frac{k}{\mu_{N,k}} e^{\frac{-x}{\mu}N-k+1}(1 - e^{\frac{-x}{\mu}k-1}) \tag{2.5.10}$$

As both PDF of Y_0 and $Z = x_{(k)}$ are known, the scaling factor T can be calculated for a required false alarm rate P_{fa} with equation 2.5.9 which results in equation 2.5.11.

$$P_{fa} = k_{N,k} \frac{(k-1)!(T+N-k)!}{(T+N)!} \tag{2.5.11}$$

It has to be noted that with this equation, the probability for a false alarm no longer depends on the mean value μ but solely on the scaling factor T the fixed value N or M describing the number of the sample cells and the sample index k. For a fixed false alarm rate the literature proposes values for the threshold T to limit the computational effort to derive these values [46].

2.5.4. SOCA-CFAR

The SOCA-CFAR algorithm is a variation of the CFAR algorithm and takes the lowest mean value of the cells right and left to the CUT [17]. Therefore, the chance to detect targets with low energy is high and the computational effort moderate. That comes with the back draw of a higher false alarm detection rate. In figure 12 the difference to other algorithms that are based on the CFAR method becomes clear with the orange box that decides in favour of the lowest mean value of x_i and y_i. This is expressed by equation 2.5.12.

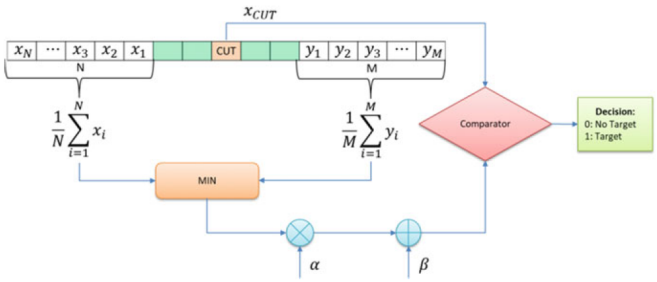


Figure 12.: SOCA-CFAR, threshold calculated by average [17].

$$Z = \min(\frac{1}{N}\sum_{i=1}^{N} x_i, \frac{1}{M}\sum_{i=1}^{M} y_i) \tag{2.5.12}$$

CHAPTER 3

Implementation

The implementation of the system is oriented at following the V-Modell. In section 3.1, the requirements of the designed system are defined. This section is followed by a description of the system architecture of the vessels GNC system and peripherals in 3.2. The part of the architecture that is further detailed in the implementation is described hereafter. The system specification in section 3.3 discusses the possible hardware, the simulation environment and the software framework used to realize the defined architecture. The software modules and its interdependencies both for the simulation and the software framework are defined in section 3.4 and are followed by a more detailed description and the implementation in section 3.5.

3.1. Requirements

This section describes the requirements that derive from the use case of the system. The requirements are separated in the three main categories safety considerations in subsection 3.1.1 and hardware and software requirements in subsection 3.1.2.

3.1.1. Regulations and safety

The design of a technical system has to be conducted by taking into account technical regulations. Both the EU and International Maritime Organization (IMO) developed regulations for the requirements of testing Maritime Autonomous Surface Ships (MASS). Among the regulations in the interim guidelines of the IMO is the demand for an adequate human-system interface and a human centred design of a MASS. Furthermore, information that is related to the vessels internal state and the data upon which the automated system judges the scene should be made available to any personnel involved in the MASS trial [40]. The regulation published

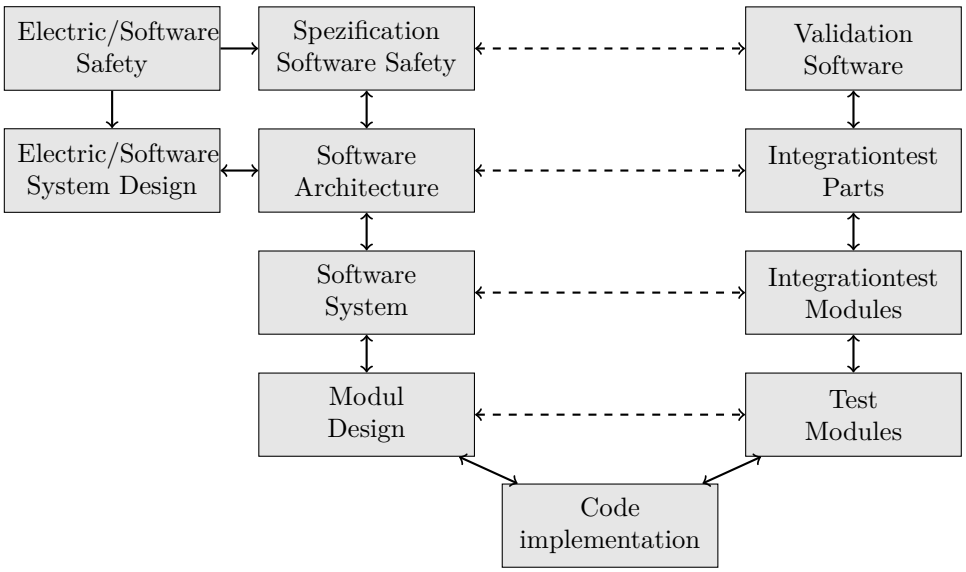


Figure 1.: Systematic development of safety related software with the V-Model after [32].

by the EU takes into consideration the aforementioned regulation of the IMO and highlights the ability of the applicant of the MASS trial to maintain a meaningful human control at any time during the test or trial with the ability to abort the trial [54].

Regarding the detailed design process, there are many organizations that develop standards for safety related software with the Institute of Electrical and Electronics Engineers (IEEE) and the International Electrotechnical Commission (IEC) considered as the most important organizations [34]. For more detailed specifications of the software requirements in programmable electronic safety related systems in Germany, the Association for Electrical, Electronic and Information Technologies (VDE) 0803-3 [33] can be acquired which is related to IEC 61508. Section 7.1 to 7.9 describe the requirements of a software safety life cycle. General information are given (7.1) and safety requirements specified (7.2). The validation of software is described (7.3) and the software development process related to safety aspects explained (7.4). The document continuous with the integration of software and electronic parts (7.5), modification processes (7.6), the software validation (7.7) and software modification (7.8) and verification (7.9). It is stated, that every model of a software safety life cycle that full fills the requirements of the document is allowed to be used with an example given by the V-Model as depicted in figure 1. The figure describes from the upper left to the upper right the states of a development process related to the safety of software and it becomes clear, that embedding this model in a traditional V-model development process is possible. First, the electrical and software safety requirements are determined to design the electrical and software system upon which the software architecture relies. Also, the specification of the software safety can be

derived from this requirements. During these steps, the software is validated and integrations tests of the software architecture are defined. The software system and module design takes place hereafter with the integration test and module test specified and applied simultaneously. The most detailed level of the design is the code implementation. The model as proposed in VDE 0803-3 is intended to be applied on large projects, therefore steps can be merged for the application in small projects.

Of special interests is section 7.4 in this document as it details the software design and development process with subsection 7.4.3 specifying the requirements for the design of a software architecture. The design of the system should be without faults in the concept, easily comprehensible and display a deterministic behaviour. The implementation of the design is required to be testable and has to be fault tolerant even in the case of external events. The responsibility for the software as well as the architecture of the design and modifications shall be documented (7.4.3.1-3). The document is continued by subsection 7.4.5 comprising the requirements for a detailed software system design. The responsibility for the software as well as the intended design of the system with related electrical parts and a plan for the safety validation of the system has to be documented (7.4.5.1-2). The software shall be programmed in such a way, that a modular, testable and changeable code is obtained and every system is divided into several more detailed submodules (7.4.5.3-4). Tests for the integration shall be specified in order to full fill the required safety integrity level. Every software code that is implemented shall be tested as specified in section 7.4.6.

It is noted, that an according regulation for the development and use of programmable electronic systems in marine applications [49] by the International Organization for Standardization (ISO) exists but is not used in favour for the more specific IEC 61508 , which is referred to in the ISO document for detailed information regarding the design process [6].

3.1.2. Hardware and software

To enable a USV to percept its environment, suitable sensors have to be found and ideally fused into a fault resistant representation of a present scene by the navigation subsystem. The navigation subsystem that integrates this sensors is required to be modular to allow the system to be updated independently from other subsystems and thus provide situational awareness in changing scenes. The exchange of information with other subsystems should be ensured and a focus has to be laid on the user of the system [20]. The resulting data of the navigation subsystem can be utilized in a subsequent step to control the USV actuators in a way that serves the objectives of the planned mission. Different sensors are proposed to be utilized in a GNC system [37]. The navigation subsystem is supposed to detect obstacles and determines its position both globally and in relation to the USV. Therefore, for long range obstacle detection, RADAR is most suitable whereas for short distances and a higher distance resolution and accuracy LIDAR serves the purpose.

To classify objects, cameras can be used. At very close range, ultrasonic sensors are an affordable solution for obstacle detection. The global position of a USV can be determined by GNSS [5]. To synchronize the mission goals with the control centre a connection to the guidance subsystem is required. Furthermore, to archive given control objectives, the influences of environmental disturbances have to be measured by a wind gauge which provides both wind direction and velocity. A water speed sensor and a three dimensional IMU serves as a feedback for the control subsystem. Ideally, the sensors can be connected to a system which allows data to be manipulated, broadcasted and send to a control station on request.

3.2. System Architecture

Figure 2 incorporates a comprehensive system diagram of a vessels GNC system with a software framework functioning as the middleware on a processor unit. The peripheral sensors are connected to the interface of the processor unit. The sensors data can be filtered and processed to such an extent that they suffice the data rate to the control centre and can be send to it via a connection of choice. A Human Machine Interface (HMI) connected to the control centre provides the possibility to send instructions to the operating system of the vessel and thus allow the full control over the system. Consequently, a connection to the actuator control enables the applicant to access relevant functions. The actuators sensors provide the control subsystem and the control centre with data of its operational status. The system architecture as depicted allows for the selection of either real sensors or virtual sensors in conjunction with a simulation environment. The simulation can be run on a different processing unit than the software framework. However, given sufficient processing power, calculations for both systems can be conducted on one system as well.

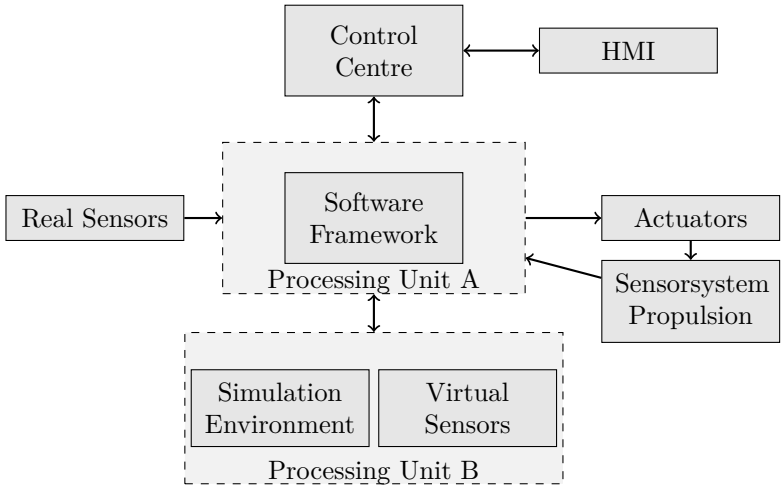


Figure 2.: System diagram of the GNC system

3.3. System Specification

Referring to the designed system architecture in section 3.2, in this section the described systems of the architecture are specified. A focus is on the peripheral devices and the processing unit, that are outlined in subsection 3.3.1, followed by the selection of a central software framework in subsection 3.3.2. Finally, the virtual environment is specified in subsection 3.3.3.

3.3.1. Peripheral Devices and Central Processing Unit

The sensors described in the following are not implemented and used in an experimental verification but its defined characteristics are important for the subsequent design and implementation of the virtual sensors. They were however chosen with respect to the appliance at hand and an implementation in a real environment is encouraged. The choice of sensors is considerably reduced by the exclusion of proprietary interfaces as these can not be connected to a third party or open source solution and thus allow only very restricted access to the sensors data. An example is given by RADAR solutions which almost always relay on the manufacturers choice of displays and therefore do not allow the data to be filtered, clustered or otherwise manipulated. Additionally a Software Development Kit (SDK) can be bought only on request, with not further specified predefined functions and executable only on the manufacturers choice of operating system. Due to this restrictions and the recent progress in solid state RADAR solutions in the automotive field [50], it has to be taken into consideration to use such a sensor in addition to a LIDAR system.

The Aptive ESR 2.5 [2] is such a automotive RADAR sensor. It applies a millimetre wave radiation, this having a wavelength of around 80Ghz, to the environment and thereby senses up to 64 targets within a distance of 174m in long range mode. In addition to the long range mode with a Field of View (FOV) of $\pm 10°$, a mid range mode with a range of 60m and a FOV of $\pm 45°$ provides a higher degree of spatial width. The Aptive ESR 2.5 can be integrated into ROS with an Application Programming Interface (API) for Controller Area Network (CAN) provided by the company AutonomousStuff. Additionally to the CAN interface which only provides data about extracted obstacles, an Ethernet interface gives access to the raw data of the RADAR. To ensure the detection of obstacles further away, the Garmin GMR Fantom 54 RADAR is used. It radiates with a power of 50W and has a range from 6m to 72 nautical miles. For closer distances of up to 100m , the Velodyne Puck [35] provides a 3D point cloud of environmental objects with 300000 points per second and for ranges of up to 7.5m, the SEN0313 Ultrasonic sensor [10] gives access to analogue distance measurements. The BFS-U3-13Y3C-C 1.3MP Blackfly camera from the manufacturer Flirr [24] allows for the classification of detected objects. The MTi-680G by XSens [3], an integrated IMU and GNSS unit, allows vessel localization and movement detection. Wind speed and direction can be measured with the WS200-UMB ultrasonic anemometer by Lufft [18]. The advantage of an

ultrasonic wind gauge solution can be found in the absence of moving parts and thus an extended lifecycle. A wireless transmission to additional devices in the network is realized by the FL WLAN 1101 by Phoenix Contact [28]. An overview of suitable sensors is given in table 3.1. In the table, the sensor type is horizontally followed by the product name of the chosen sensor and an estimated price as well as the interface the sensor uses to transmit data.

To integrate the sensors and connect its data, a central hardware solution has to be found. The Jetson AGX Xavier hardware specifications qualify it as the central data processing unit. The Xavier Development Kit provides a development platform with many standard hardware interfaces that results in a highly flexible and extensible platform for rapid prototyping. It uses a maximum power of 30W for its own supply and can supply external peripheral devices with an overall power of 35W. The Xavier Development Kit can be interfaced via USB, UART, I2C, CSI-2, Ethernet and a variety of other ports. It is run by a derivative of the operating system Linux Ubuntu. As the hardware platform hosts a powerful GPU with according libraries such as OpenCV and TensorRT and a Multimedia API, it is a suitable solution for sensor integration and data processing [8].

Table 3.1.: Sensors for environment detection

Type	Name	Interface
RADAR	Aptive ESR 2.5	CAN/Ethernet
RADAR	GMR Fantom 54	Ethernet
LIDAR	Velodyne Puck	Ethernet
IMU		
GNSS	MTi-680G	CAN
Velocity		
Windspeed	WS200-UMB	RS485
Camera	Blackfly 1.3 MP	USB 3.1
Processor	Jetson AGX Xavier	various
Ultrasonic	SEN0313	digital
WLAN	FL WLAN 1101	Ethernet

3.3.2. Software Framework

To integrate the sensor and provide a middleware to execute the GNC system, the frameworks ROS, Control Architecture for Robotic Agent Command and Sensing (CARACAS) and MOOS-IvP are taken into consideration. These system platforms facilitates the effort to program the required functions with predefined libraries and communication structures.

- The CARACAS architecture is available on request for research projects and is potentially applicable on diverse robotic systems. The system is divided

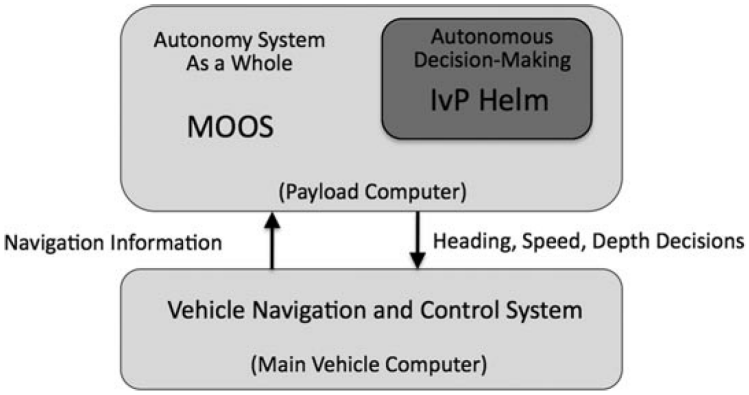


Figure 3.: System diagram of the MOOS-IvP architecture after [4]. The navigation and control system is physically decoupled from the autonomy system and is not further specified.

into three agents that interact with each other and a data storage that hosts a world model. The objective of the agents is to provide the robotic system with the capabilities of both deterministic reactions to unanticipated occurrences and re-planning in the event of changes in goals and resources. The focus therefore is on the guidance subsystem of the GNC system [52].

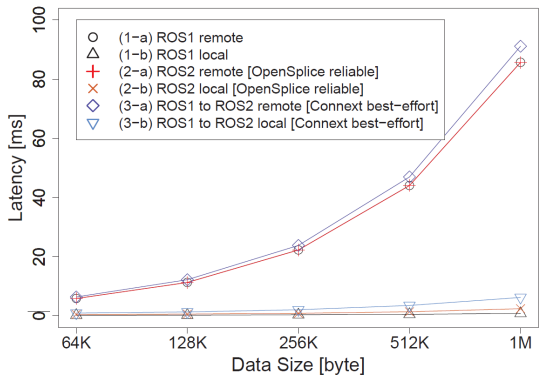
- MOOS-IvP is an open source project and specifically intended for the application in unmanned marine vehicles. An autonomy system, which can be referred to as the guidance subsystem, runs the autonomous decision making helmet MOOS-IvP which is connected with an input to the data of a navigation subsystem and with an output of data of the control system (figure 3). Therefore, it is decoupled from the platforms hardware and does not require any specification of how the navigation and control subsystem is implemented [4].

- ROS is an open source software which is developed with the objectives of modularity, usage of a peer to peer structure and being based on tools. A low level device control provides drivers for peripheral devices and a hardware independent architecture allows ROS to be run on a wide variety of devices. Community based libraries provide methods for sensor data processing and visualization, obstacle detection, mapping and path finding [44]. As the sensor integration into the GNC system is vital in the design of a new robotic platform, ROS can be evaluated as a middleware software.

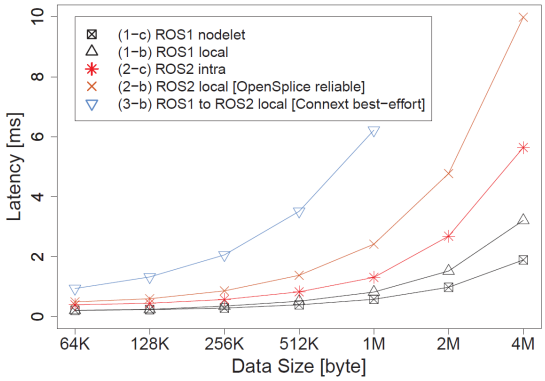
With two different implementations of ROS, namely ROS1 and ROS2, the choice has to be made which of them is best suited to the application at hand. Developed since 2007, ROS1 provides a comprehensive set of packages [31] and is supported until 2025. Therefore, four more years of active development is ahead of the current long term supported derivation, which is *Noetic Ninjemys*. With the four years ending, adding new hardware can proof difficult because of the lack of drivers, however

the system as developed before would continue to be functional. ROS1 is not build with native real time support [43]. Released with an first beta version in 2016, ROS2 still needs resources to implement interface functions and migrate packages from ROS1. For example Unity features are not supported at all and Gazebo is supported only partially in ROS2. However, ROS2 provides real time characteristics (note that real time capability depends on the underlying operating system as well as on the device driver [19]), is suitable for small embedded platforms and can be run not only on Linux. It implements a Data Distribution Service (DDS) in exchange for the ROS1 message transport system. DDS follows an industry standard and provides real time communication by various configurations such as deadline, reliability and durability specifications [38].

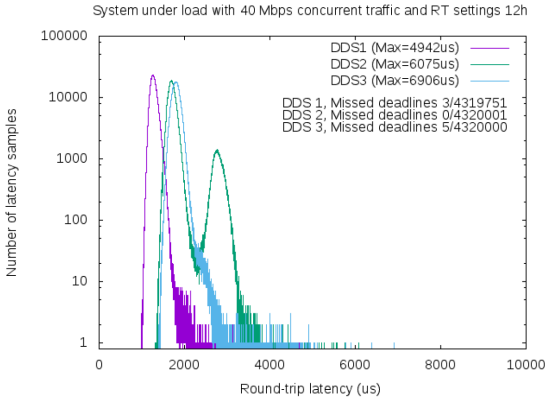
To further verify ROS as a middleware for interfacing sensors and process its data, the performance of the software in situations with a high amount of data traffic has to be evaluated. In figure 4, the data latency [ms] with respect to the data size [$byte$] is plotted. Two machines with an Intel Core i5 are used for remote and local transmission of a string typed message in a 10Hz interval. The latency difference for data of 4Mb in figure 4a for ROS1 Indigo and ROS2 Cement can be neglected in comparison to the difference of remote and local data transmission with a latency of about 80ms and 5ms respectively. Thus, when dealing with large datasets as resulting from for example LIDAR point clouds, processing data locally is to be preferred over the transmission of data to another data processing device. The performance of local data transmission in ROS is detailed in figure 4b. Performance of data transmission by a pointer (referred to as nodelet in ROS1 and intra in ROS2), by a node via Transmission Control Protocol (TCP) (referred to as local), by the OpenSplice implementation of DDS set to reliable and in between ROS1 and ROS2 are compared. The performance of a nodelet in ROS1 exceeds the other transmission modes with a latency of less then 2ms at a data size of 4Mb. It is followed by a ROS1 node with less than 4ms. The performance of the communication between ROS1 and ROS2 with 5ms latency at only 1Mb of data size is the least performing transmission method [38]. As for a real time system, not only the transmission time, but the compliance with predefined timing deadlines is important, in figure 4c and 4d the number of messages that missed its deadlines are displayed. The tests are conducted with two different machines that run Linux with a PREEMPT-RT patched kernel, which is a real time kernel implementation for Linux. The Round Trip Time (RTT) from one machine to the other are measured with the ROS2 profile "best-effort reliability" and three different DDS implementation in each test. In figure 4c the message latency is plotted against the number of samples with that latency. Even with a system traffic of 40Mbps and the processor under load, three, zero and five samples in accordance with the DDS preferences missed the preset deadlines over a test time of 12h and of a total of 4320000 samples. In figure 4d the same test was conducted but only over a time period of 10 min and with an increased traffic of 80 Mbps. This results in missed deadlines of 1470, 1799 and 570 missed deadlines dependent on the DDS settings with a total of below 60000 send



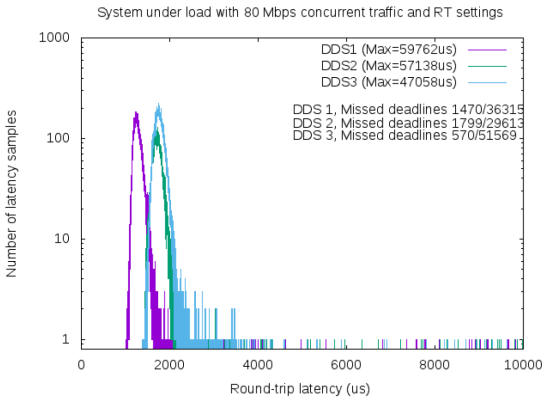
(a) Latency in ROS1 and ROS2 in $[ms]$ for remote/local data transmission of 64 Kbyte to 1 Mbyte by [38].



(b) Latency in ROS1 and ROS2 in $[ms]$ for local data transmission of 64 Kbyte to 4 Mbyte by [38].



(c) ROS2 round-trip latency in $[\mu s]$ for three different timing deadlines for 40Mbps in 12h under CPU load by [19].



(d) ROS2 round-trip latency in $[\mu s]$ for three different timing deadlines for 80Mbps in 10min under CPU load by [19].

Figure 4.: ROS data transmission performance.

samples. This can be considered as not sufficient for real time appliances [19].

It can be concluded, that in having a supported deterministic behaviour and a long support window, the advantage of ROS2 outweighs the disadvantage of lacking development resources in the appliance to the middleware of the GNC system. In addition the interface to the game engine Unity3d is proven to work for ROS2 as well as ROS1 with only small changes in the workflow. The drawback of a less extended development community is presently becoming smaller as the industry more and more supplies software and support for ROS2. This is a trend that can be expected to grow in the future. Therefore, ROS2 is chosen as the middleware for the implementation of the GNC system.

3.3.3. Virtual Environment

Virtual environments has been long used in the development of robotics to test new concepts, strategies and algorithms [30]. With the inherent limitations of a real world testing environment such as unpredictable weather phenomena, wind induced currents and wakes and time dependent changes of light, simulated environments can increase the availability of suitable test variables, to name one advantage. There are many software solutions to simulate environments, however, especially in the field of USV, using the simulation software Unity3D is a viable choice and can be preferred to other simulation software [58]. Unity3D is a game engine developed by Unity Technologies and allows for easy development of interactive scenes with a high visual fidelity. An asset store in which a large community publish textures, models and extensions and other types of predefined assets makes the development process of a new scene efficient. Compared to most other simulation environments, the elements of the virtual environment built with Unity3D are close to the actual environment with an example given by an available high fidelity virtual water surface environment. Furthermore, virtual sensors can be modelled and embedded into the scene through scripts [58]. Unity features a powerful PhysX physics engine, a render engine and collision detection engine among others. Of special interest is that there is available a library called *rosbridgelib* for Unity, that links the architectures of ROS and Unity by using a JavaScript Object Notation (JSON) data format to exchange messages [22]. It has to be noted that this library is officially only implemented for ROS1 and planned only in 2022 for ROS2 as of information in 2021. However, as the message exchange format between ROS1 and ROS2 only differs in specific use cases, the library can as well be used for ROS2.

3.4. Module Design

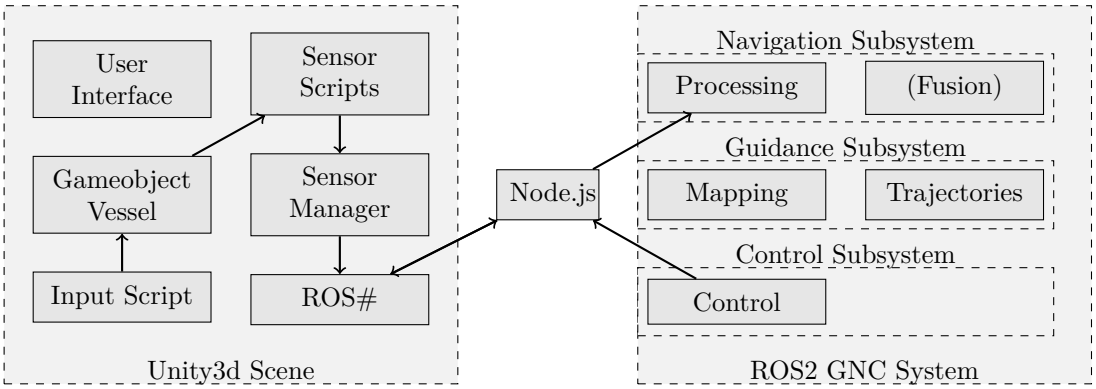


Figure 5.: Modules of the proposed system

With the tools set according to the requirements, the module design takes place as depicted in figure 5. With a focus on the virtual environment, physical sensors

are left out of the figure but can as well be integrated in conjunction with or in place of the virtual sensors. Figure 5 is divided into two main components that are connected with the ROS webbridge which is implemented in Node.js. Node.js is an asynchronous event-driven JavaScript runtime environment with the purpose of building scalable network applications. The Unity3d Scene to the left of the Node.js component is composed of a User Interface with which preferences to the Unity Scene such as the screen resolution can be chosen. An input script enables the user to send steering information via keyboard to the virtual vessel which is a Unity3d Gameobject and thus the position of the vessel can be manipulated. Assigned to the vessel are the scripts that simulate the sensors. The scripts send its data to a central Sensor Manager where the data structures are defined with which the data of the sensors can be send with the ROS# to the ROS webbridge. With the introduction of a Sensor Manager, sensors of different types can be easily integrated into the scene and its data types manipulated without changing definitions at various places inside the Unity3d Scene editor.

With the virtual environment set, the ROS2 component right to the Node.js component is required to receive the sensor data and manipulate it. This is done in three main categories of functions that are set up according the the specification of a GNC system. First the Navigation Subsystem processes the received data. As the sensor data, particularly when transmitted combined, can amount to a high data rate, it is advantageous to process them directly in the procedure that receives the data. After processing the data by filtering, feature extraction and clustering, the results of the different procedures can be fused into a fault resistant representation of the scene. With this representation, the Guidance Subsystem creates a map and calculates trajectories. The points prescribed by the trajectory are then used by the Control Subsystem to generate steering information either for a physical or a virtual implementation of the vessel. As required, the results of the processes inside the categories have to be visualized to the user, so that decisions of the system are understandable.

3.5. Module Implementation

This section implements the modules that are specified in section 3.4. In subsection 3.5.1 the scene in the Unity3d game engine is modelled with static elements such as the terrain and objects. This is followed by the design of the dynamic elements that are navigated through the scene e.g. a vessel and different sensors (3.5.2). A third subsection 3.5.3 details the ROS network that is programmed to process the data of the sensors in the Unity3d scene.

3.5.1. Static environment

In a first step, the virtual scene is modelled in Unity3d running on Linux 20.04 LTS. As a geographic reference for the terrain, the location proposed for the physical implementation of the project ELLA is used. Elevation data for this location in the reference coordinate system EPSG 25832 for the position and EPSG 7837 for the height from east 384999 to west 383000 and from north 5720999 to south 5719000 is provided in .xyz data by OpenDataNRW with a resolution of one meter [15].

The data is loaded into to the software QGIS Version 3.16.6 LTR. QGIS is an open source geographic information system and provides various functionalities in this domain. One of them is the creation of a Triangulated Irregular Network (TIN) which can represent a Digital Elevation Model (DEM). Given a function $z = f(x, y)$ in which x and y represent the position data and z the according height, the function can be computed only for discrete points as x and y are finite in S and depend on the resolution of the source. Therefore, in order to map the DEM continuously, a TIN has to be created that connects the discrete points so that they suffice three conditions. First the number of vertices T of the resulting triangles have to be equal to the number of points S. As a second condition the interiors of the triangles out of T are not allowed to intersect each other and if the boundaries of two triangles intersect each other, the intersection has to be either a common edge or a common vertices. On the resulting triangles, a linear interpolation function can be applied and thus a continuous DEM approximation is achieved [16].

To convert the DEM to a .raw heightmap that is used by Unity3d to create a terrain, the DEM is converted to a .TIFF and exported to Gimp. Gimp is an open source image manipulation program and supports various file formats of which the .raw format is one. The resulting image is shown in figure 6. On a 256-bit greyscale resolution the darker regions of the image represent low terrain with increasing height for every step towards white. With the import function of Unity3d the terrain is then shaped according to the heightmap. The resulting terrain data is visualized in figure 7. The sink in the bottom-left of the image matches the second dark shape along the top of figure 6.

Another option to create a terrain out of the DEM is represented by importing

Figure 6.: Height map of the scene that is to be modelled

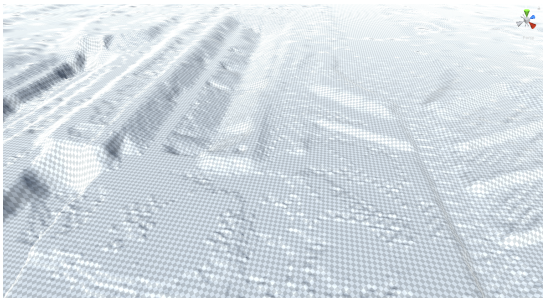


Figure 7.: Terrain data in Unity3d derived from a height map

the data into blender and create a mesh that is then exported to Unity3d as an object. However it turned out that the advantage that comes with the possibility of granular mesh manipulation is at the same time a disadvantage. A highly intersected mesh with a very large amount of vertices can not be easily joined without destroying the integrity of the mesh, that is the consistency of vertices connections. Thus the larger the area that is modelled and the more details in terms of position resolution is required, the less manageable the mesh becomes in terms of processing time required for visualization.

The terrain as shown in figure 7 is smoothed which is necessary because of the limited resolution of the height map that is restricted to 256-bit on the greyscale that represents the height. On the resulting terrain, a satellite image of the scene serves as a background texture. Various other textures are then applied to the scene to resemble the real environment. In the software Blender 2.92.0, 3D objects are created and imported into the virtual environment. As the process of creating the objects and applying textures and shader as well as collision meshes to the objects is rather tedious, only objects that are vital to the scene are modelled. For reasons of simplicity only rigid bodies are used in the scene. Rigid bodies are volume bodies in which every set of two different arbitrary points contained in the body have the same distance to each other at any time. Figure 8 shows the Blender model of the small vessel that is used in the Unity3d scene. It is composed of multiple connected objects that can be used to derive a collision mesh from. The collision mesh surrounding the vessel inside the modelled scene is depicted in figure 9 with the mesh coloured in green. There are two option for the generation of the mesh in Unity3d. A concave mesh takes the vertices inside the object and a convex mesh is laid over the vertices outside. The collision mesh as depicted is a convex shape. It can be noted that there are differences of the mesh to the shape of the visible object that results of the inability of the mesh to project on concave and convex parts in the same object.

The scene as seen from above is shown in figure 10. The V-shaped river is in the center of the image with a total of five landing stages and two small vessel located

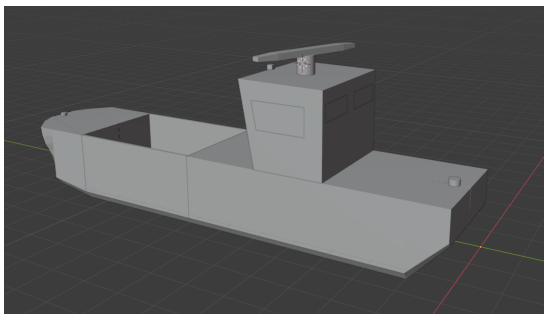


Figure 8.: Blender vector model of a small vessel used in the modelled scene

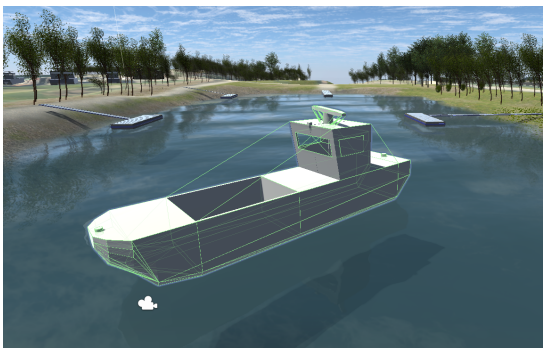


Figure 9.: Small vessel with collision mesh generated from the blender model

on it. With a satellite image as a background, texures for paths, meadows and the rivers shore are applied on the terrain. Trees and houses further detail the scene. The water is represented by a mesh of Unity3d's standard assets. The standalone application that can be run without a Unity3d IDE can be seen in figure 11. Settings in the top left of the application allows for the change of perspective out of which the scene is rendered, the change of resolution and for reloading or closing the scene. In the foreground the small vessel on which the sensors are mounted can be seen. In the background, the aforementioned textures and objects are visible.

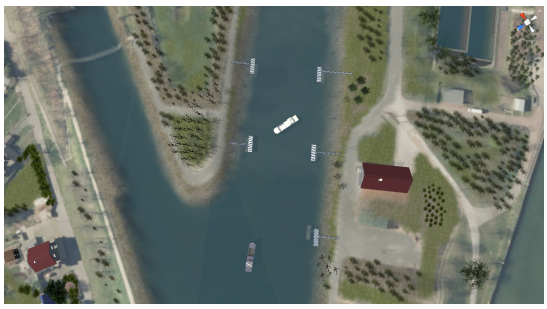


Figure 10.: Unity3d scene from above

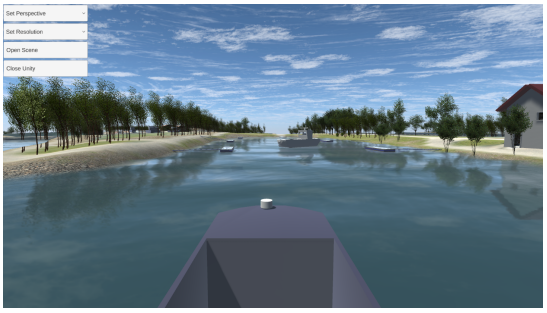


Figure 11.: Unity3d with menu from sensor perspective

3.5.2. Dynamic Environment

With the scene modelled, defining its behaviour during runtime is the next step. To obtain a realistic physics simulation, Unity3d makes use of a physics engine. The main task of a physics engine is to solve forward dynamics, however there are many factors that determine the performance of a physics engine and thus there are many different implementations available. Collision and energy restitution, the modelling of constraints and the integrator performance are among the factors that can be evaluated in order to determine a suitable implementation. Unity3d default physics

engines are PhysX supplied by Nvidia and Havok Physics supplied by Havok. It has to be noted, that both engines are optimized towards the appliance in games and therefore a strong focus is on the engines stability rather then the accuracy [13]. In Unity3d, the physics module provides an abstraction level to access functions related to 3D physics. These functions are used among others to steer a small vessel with a keyboard input through the scene. The forward movement and the rotation of the vessel are taken as an input and serve the purpose of altering the position of the virtual sensors that are mounted on the vessel.

All data types of the sensors are saved in a central script called Sensor Manager. For every sensor instance, a public data storage class is instantiated that can be accessed by the sensor class as well as by the class that sends the data to the ROS network. The foundation of what is used as a LIDAR sensor and IMU sensor in Unity3d is already defined. Only the saving process is changed to allow an adaptation to the outlined data exchange method. The LIDAR sensor class makes use of the so called raycast, which is one method provided by the physics engine to provide scene queries. A point cloud in the scenes global coordinate system results of the colliders that are hit by the emitted rays. Various variables such as the field of view and the sweep frequency allow for the simulation of physical implemented sensors. To transmit the pointcloud, its three-dimensional cartesian positional data is saved with one list for each dimension. A list with an altering size for each cycle comes with the disadvantage of a required memory allocation for every measurement cycle. However, as the data size of the pointcloud is changing depending on the objects that are hit by the raycast, a list allows for having the lowest data size possible.

Based on the physical operating principle of a real RADAR, a virtual RADAR is implemented in Unity3d as depicted in figure 12. The emittance of an electromagnetic wave of a real RADAR is simulated by a discrete beam that consists of raycasts. As a sequential calculation of the collisions of objects with the emitted rays increases the process time and decreases the simulation performance, a job handle is used that is implemented in the LIDAR sensor as well. A job handle allows for parallel execution of tasks and thus can increase the process time. The resolution of the discrete beams is adaptable with a variable in the script of the sensor. Each time an object is hit by the raycast, the positional data is saved together with a reflectance value of the object. The reflectance value is provided by a script that can be attached to objects to further detail its material properties. That is useful especially for a RADAR as the reflected energy not only results of the object size but of the objects material, too. After one turn that consists of a predetermined number of beams, the cartesian three-dimensional positional data of all detected objects is mapped to a two-dimensional space by omitting the height data.

To resemble the image of a radar, an array is set up with the length resulting of the predetermined number of beams multiplied by the set cell extension in every beam. Starting at the center M that is the sensor center, the array consists of the

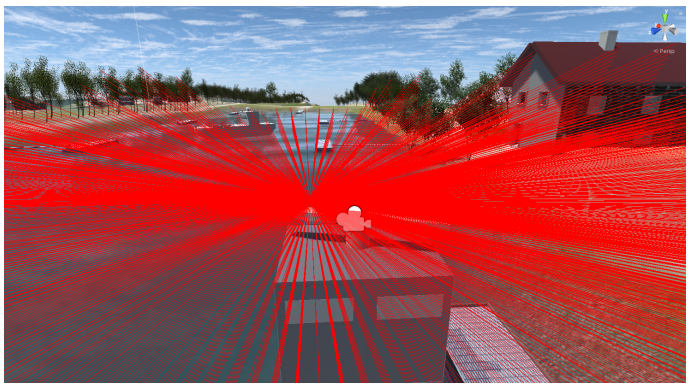


Figure 12.: Raycast of the radar with active rays visualized in red

subsequent beams cell values. Each cell can be accessed with an index as shown in figure 13 and is defined by its radius r_{min} and r_{max} and angle theta t_{min} and t_{max}. The quantity C of points inside the cell now follow equation 3.5.1.

$$C = r, t | r_{min} < r > r_{max} \wedge t_{min} < t > t_{max} \tag{3.5.1}$$

The two-dimensional cartesian positional data is transformed to a quantity of polar coordinate representations called P. The points I inside the cell now follow equation 3.5.2.

$$I = P \cap C \tag{3.5.2}$$

The reflectance value r of the each cell can be calculated with equation 3.5.3 with ref_i being the aforementioned individual reflectance for each object. Each value can be saved to the array that can then be transmitted to the Sensor Manager.

$$r = \sum_{i=0}^{i=|I|} I_i * ref_i \tag{3.5.3}$$

To send sensor data to ROS, a library for Unity3d called ROS# is used. ROS# can be integrated into Unity3d and makes use of the messages that are defined in ROS1. As there are only few differences in the default messages provided by ROS1 to the subsequent implementation ROS2, the library can be used for both frameworks. However, it is noted that the message of type header makes use of a different notation for time units and thus has to be adjusted. In addition to default messages, ROS provides the possibility to define custom messages that can then be exported to ROS#. Custom messages in the context of the particular use case are derived from the sensor data classes. As a webbridge run by Node.js serves as an intermediary framework for data transmission, new messages types have to be introduced to it as well.

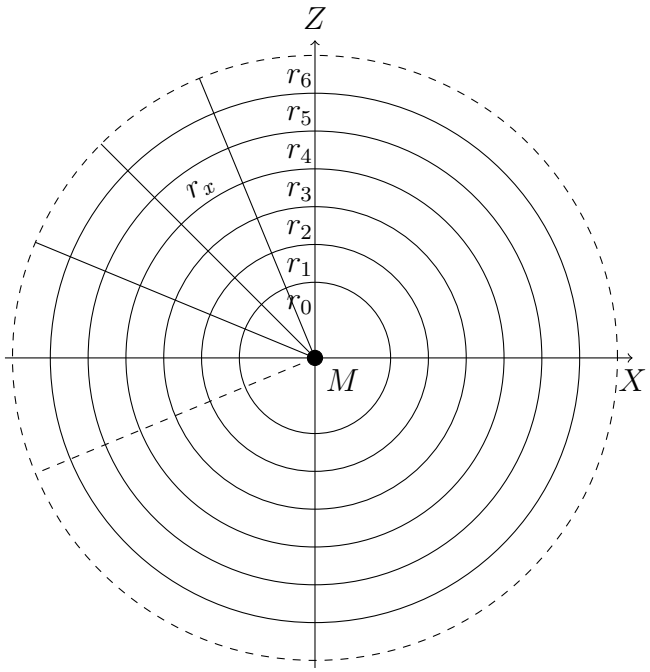


Figure 13.: 2D Cartesian and polar coordinates with index and reflectance value of RADAR cells.

3.5.3. Data processing

New message transmitted to the ROS network are first received by the nodes that process the message data. In ROS nodes are executables that can communicate to other nodes with an interface such as datatypes defines in messages. To avoid stressing the network with high data rates, for large messages a design maxim in this project is the reduction of data size in the first node that receives the data. That can be done by algorithms that for example filter, cluster and segment data.

Figure 14 displays the structure of the data processing in ROS. The node transform_listener is not used as it is only relevant for coordinate system transformations that are not taking place in the scope of this work. The ros2_web_bridge-node connects to the programmed Unity3d scene and receives its messages. A total of four ROS nodes that are named image_subscriber, lidar1_xyz, radar1_img and get_imu process the incoming data. After being processed, the lidar and radar data are published again and are received by the n_rviz node that is a predefined ROS visualization tool.

The incoming IMU data is not further processed but solely displayed as shown in figure 15. The angular velocity in x,y and z as well as the acceleration information of the sensor is plotted in a time frame of five seconds. As in Unity3d the sensor is mounted on a object without further specified physical characteristics, the object

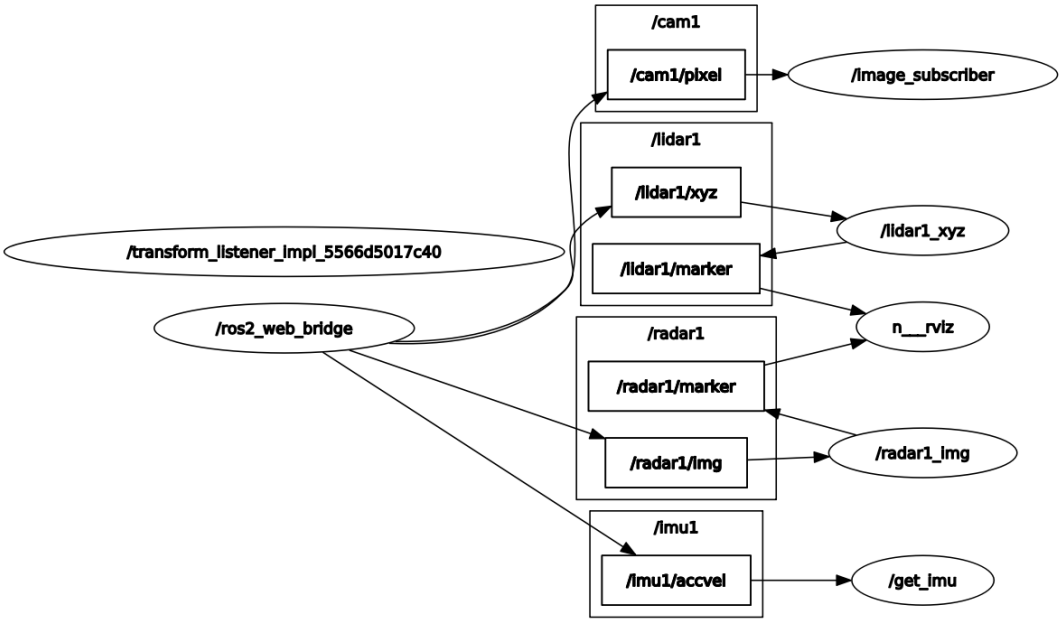


Figure 14.: Node and interface structure of ROS2 navigation package

acceleration in positive as well as negative direction is not further specified by properties such as weight. Presumably this accounts for the rapid changes in for example the acceleration in x direction. The visualization makes use of the *matplotlib* library for python with the plot style set to continuous plotting. It has to be kept in mind that the library as well as ROS has its own event handling and resulting loop. Therefore, not every setting of *matplotlib* can be used for plotting data together with ROS.

The LIDAR data that is received from the Unity3d scene has to be processed in order to extract obstacles that are used to map the environment. As there are many algorithms that operate on large datasets of three dimensional data, the approach that is best suited for the application has to be found. The Hough Transform Algorithm directly extract contours in the data it is applied on. That comes in handy as it requires no other processing. However as on every point of the data iterations are conducted, it can be expected that the method does not provide the performance used for realtime applications in which new data comes in at high intervals. The same drawback counts for the disqualification of the partition based K-Means clustering algorithm. Additionally, this algorithm only gets to local optimal solutions in a limited processing time. The hierarchic BIRCH clustering algorithm tends to subdivide large cluster that in fact are intended to be viewed as one cluster. The last discussed algorithm is the density based DBSCAN clustering algorithm. An advantage of this method is the detection of noise that can then be neglected for mapping the environment. This algorithms promises the best results as no disadvantages can be noticed. Thus it is applied in the node that clusters the LIDAR data in an implementation that makes use of parallel procession of points [55]. Fig-

ure 16 displays the segmented data. For example the coloured vertical lines can be classified as colliders of trees. Again, *matplotlib* is used for the data visualization.

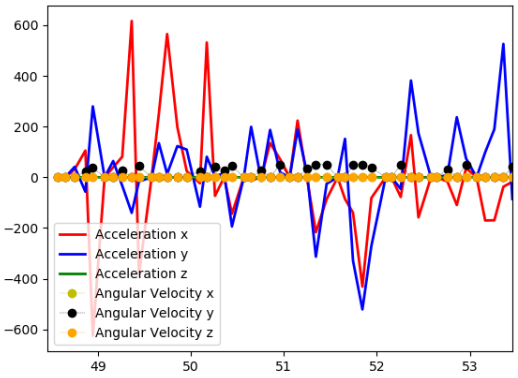


Figure 15.: IMU data with acceleration and angular velocity

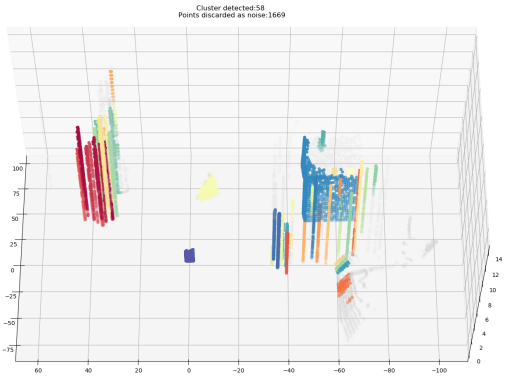


Figure 16.: Lidar pointcloud with colored clusters

As by clustering, the datasets size is not reduced apart from the neglected noise, the resolution of the clusters has to be reduced. That can be done by finding one or more shapes that enclose the cluster and thus includes all of its data. For reasons of simplicity only rectangluar shapes with a fixed orientation are used. In figure 17 the black circles represent a two-dimensional cluster. The smallest rectangular shape with a fixed orientation parallel to $X$and Y that includes all data points is the box that is set up by the lines a, b, c and d. Being required to run in parallel to Y the lines a and b are defined by the outmost position of points to the left and to the right respectively. Lines c and d run in parallel to X and are defined by the highest and lowest point respectively in respect to the Y axis. By dividing the resulting box into four even sized smaller boxes, the resolution of the cluster can be increased. As depicted in figure 18, boxes that are not populated by points of the cluster can be discarded. Applied on the three-dimensionl lidar cluster, the boxes are intersected as long as they do not full fill the condition of being smaller than three meters in each dimension. For every resulting box it is checked whether there is a point inside the box. Boxes that contain points are then send to the ROS network by taking their center and the extension in every dimension.

After receiving a RADAR array with reflection values from Unity3d, its data is visualized in a *matplotlib* plot in polar coordinate representation. Figure 19 displays the resulting image. The byte typed array of reflection intensities is plotted for every cell from 0° to 360° with the according radius. To decrease the false alarm detection rate, a dynamic threshold is used in a next step to extract targets. The array that contains the detected signals is computed back to carthesian coordinates in order to obtain a general data format that was introduced by the LIDAR data. It then is

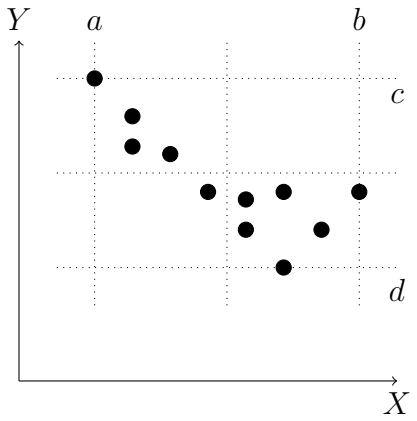


Figure 17.: Possible boxes to include data points of one cluster

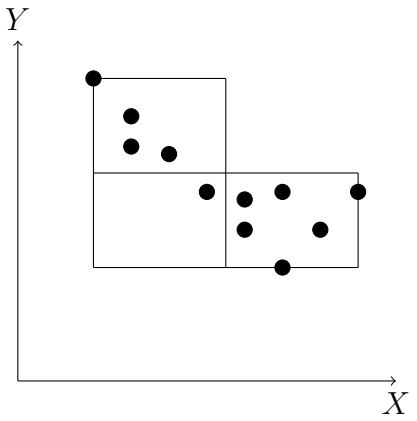


Figure 18.: Boxes resulting of discarded empty boxes of the cluster

send to the ROS network by taking the center of each detected signal.

In the scope of this work, only a two-dimensional collision mesh for the simulation of water is used and thus there are no wakes that can be falsely classified as a signal. Therefore the implementation of a OS-CFAR algorithm is not required and the SOCA-CFAR is used. However, in more sophisticated simulations or real world applications where situations can be expected in which many objects have to be detected that are in close proximity to each other, the OS-CFAR is the algorithm of choice. Additionally, with the ability to model the strength of wakes with the Rayleigh PDF, the OS-CFAR provides further chances to reduce the false alarm rate.

Especially in artificially confined spaces with defined borders an algorithm that detect lines in the scenes images can provide useful information to specify the environment. In order to detect lines, the edges of the image have to be extracted. For that the python library OpenCV with a focus on real time applications for computer vision provides the so called Canny Edge Detection function. To extract edges, first a Gaussian filter is applied on the greyscale image to reduce noise. In a next step the edge gradients in vertical and horizontal directions are computed. The function allows for choosing the number of pixels that are included in the calculation for each gradient. If the gradient value is above a threshold one, the position is considered as an edge. If its below a threshold two, the value is discarded. For the values in between the two thresholds, the decision whether an edge is present or not is dependent on the existence of values above threshold two in the neighbourhood. The resulting edge image can be passed to the Hough-algorithm which then extracts the lines. Figure 20 visualizes the result of a line extraction applied to the image from the image sensor. Three blue lines indicate the detection of lines from an image, that has been passed to a edge detection beforehand. As the serialization of bytes is not implemented in the *rosbridgelib* that connects Unity3d with ROS2, the im-

age is saved in the underlying Linux filesystem. Only a message that indicates the availability of a new image and the place where it is saved is published in the ROS2 network. After the according ROS node receives the message, the image is read from the specified storage place and can be processed.

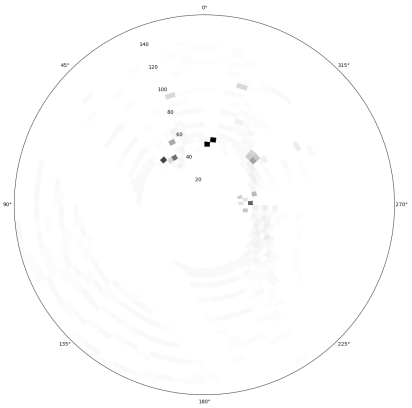


Figure 19.: Radar data with reflection intensity

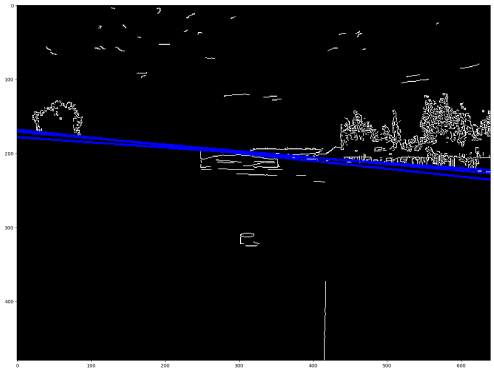


Figure 20.: Edge image with visualized hough line detection in blue

As a last step, the processed information of the LIDAR and RADAR are displayed together. Figure 21 displays the RADAR data in red circles together with the LIDAR data in yellow rectangles from a perspective above the scene. Figure 22 displays the same state of the scene from a third person perspective. The aforementioned bounding boxes of the LIDAR data become visible in yellow with the size of a single box not exceeding a certain limit. The sensor center is indicated by small coordinate system arrows in the center of the figure. Around the arrows the shape of the small vessel is displayed. The two dimensional RADAR data is visualized with red cylinders.

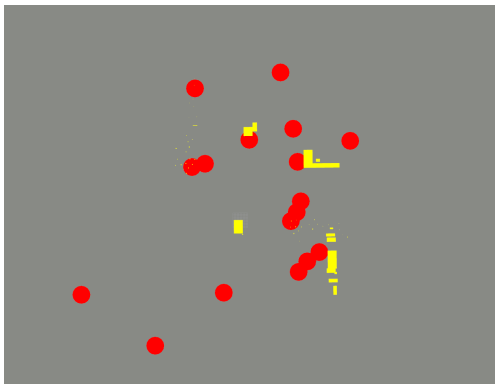


Figure 21.: Lidar (yellow) and radar (red) objects from above

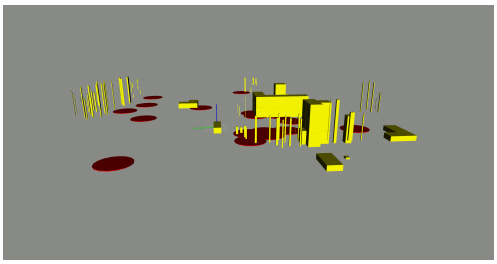


Figure 22.: Lidar and radar objects in third person perspective

CHAPTER 4

Results and Discussion

With the suitable algorithms implemented, in this section the parameters of the algorithms are further discussed. As the data of the virtual sensors are dependent on its preferences in the Unity3d scene, they are documented in subsection 4.1. Subsequently the performance of the system is evaluated by taking the latency of every processing step. The tests are strongly focused on the median latency and reliability in terms of timing inaccuracies and scattering of the data generation, transmission and processing. The tests that are conducted in the following can be divided into the test of the modules which goes along with the integration of the modules in subsection 4.2, the test to proof the successful integration of parts in subsection 4.3 and the validation of the software as a whole in subsection 4.4.

4.1. Sensor Parameters

The virtual sensors as implemented in the Unity3d scene have to generate data about its digital environment by reproducing the physical properties of real sensors. Inherent to this approach is the limitations that are set by the processing power of the unit that calculates the virtual sensors data. Therefore, in addition to the values that can be derived from real sensors, the performance of the system has to be observed when for example increasing the resolution of a sensor to resemble an analogous sensor signal by means of discrete digital values. The settings of the sensors and data processing algorithms values are determined experimentally and merely provide an example for the possibilities of the overall system architecture. Therefore in the following the validation of the values is conducted on a qualitatively basis without comparing the performance effects of different sensor settings.

The resolution of the virtual RADAR is an example for the necessity to deviate

from the real sensor as proposed before. To realize an object detection in a range of several kilometres with a detection rate that equals that of an analogous RADAR, the digital RADAR's angular resolution exceeds the processing power of a common computer. Thus its detection range is limited to about 100m for objects below four meters in the current implementation. Table 4.1 provides an overview of the settings of the RADAR. One beam of the RADAR is five degrees wide in azimuth and 15° wide in altitude. The resolution of one beam is 30x2 raycasts. For the detection of objects close to the RADAR's origin the resolution proved to be sufficiently detailed. However, after exceeding a certain range, small objects are able to pass the RADAR's rays undetected due to the increasing space between subsequent rays. To increase the detection rate of objects, providing them with a very high reflection value of about 100 provided good results when applying the SOCA-CFAR filter.

The settings of the filter that proved to be most effective are documented in table 4.2. A total of 100 cells are taken into consideration to compute the decision whether the CUT is classified as an target object or not. Ten cells that are five to the right and five to the left of the CUT are ignored and thus decrease the chance that signals of objects are taken into consideration when computing the noise average. The false detection rate is set to 0.1. In contrast to a fixed threshold, the SOCA-CFAR provides usable results even when changing the settings of the virtual RADAR which for example results in an increased overall reflection value. That can be considered as an effect of the dynamic threshold calculation.

Table 4.1.: RADAR sensor values

Unity3d	Value
Cell Azimuth Extension	5.0
Cell Extension	4
Cell Altitude Extension	15
Burst Vertical Resolution	30
Burst Horizontal Resolution	2

Table 4.2.: Filtering values

SOCA-CFAR	Value
Training Cells	100
Guard Cells	10
False Detection Rate	0.1

The physical principle of operation of the LIDAR allows virtual data generation without further transformation and approximation. Thus, the data generation in the simulation can forgo without substantial compromises. As listed in table 4.3 the LIDAR is set to about 345000 points with a frequency of two Hertz. The horizontal and vertical resolution are 0.25° per ray with an field of view of 360° and 60° respectively. The procession of the points with the DBSCAN algorithm yields the most promising results with the settings in table 4.4. The values have been determined experimentally with adjustments taken in comparison with the objects spatial limitations as visible in the Unity3d scene. An *Eps* value of one equals to points being considered as neighbours with an distance of below one meter to

each other. *MinPts* set to ten indicates that a minimum of ten points is required in a cluster to be considered as such. The resolution of the bounding box is a further value that can be set. With applications prioritising accuracy, the resolution can be set higher and thus allow for boxes that frame the data more precisely. However, in situations that require a highly responsive system, compromising the accuracy by decreasing the resolution to higher bounding box values can lead to a higher performance in terms of processing time. The value of three that equals to three meters provides a sufficient accuracy without using to much processing time. By choosing the resolution and compression of the rendered image from the

Table 4.3.: LIDAR sensor values

Unity3d	Value
Horizontal Resolution	0.25
Vertical Resolution	0.25
Vertical Field of View	60°

Table 4.4.: Clustering values

DBSCAN	Value
Eps	1
MinPts	10

Bounding Box	Value
Resolution	3

Unity3d scene, the data size of the visual environment information can be reduced or increased. As shown in table 4.5, 640 pixel in width and 480 pixel in height are chosen as values for the image resolution together with a jpeg compression. These values provide enough details for the subsequent line extraction and at the same time allow for an acceptable procession time. For the edge detection and line extraction, table 4.6 provides information of the used variables. The algorithm for the edge detection as proposed by Canny uses 170 for a first threshold and 260 as a second threshold. With an value above one for *Rho* as the distance in pixels between sampled pixels and a *Theta* of one as the angular distance in which lines through edge points are computed, the Hough algorithm provides a sufficient computation time. To decrease the false alarm rate, the *Threshold* is set to 190. Increasing this value results in a higher amount of lines with the same value in the Hough feature space for an edge to be considered a line in the Hough image space.

4.2. Module and Module Integration Test

The specifications of the hardware that is used to test the system is given in table 4.7. With 30 GiB of random access memory, 16 cores at 3.60 GHz and a RTX series graphics card, the system can be considered as being up to date. As an operating system Ubuntu LTS is used. This Linux derivative has the advantage that it is supported by Roboter Operating System 2 (ROS2) as well as Unity3d.

Table 4.5.: Image sensor values

Unity3d	Value
Width	640
Height	480
Compression	Jpeg

Table 4.6.: Line extraction values

Canny	Value
Threshold 1	170
Threshold 2	260
Hough	**Value**
Rho	1.1
Theta	1°
Threshold	190

Table 4.7.: Hardware used for testing

Category	Type
Processor	Intel Core i9-9900KF CPU @ 3.60GHz x 16
Random Access Memory	30GiB
Graphics Card	Nvidia TU104 GeForce RTX 2070 SUPER
Operating System	20.04.2 LTS Ubuntu

To test the modules that are represented by methods in Unity3d and ROS2, the time each method takes to be executed is saved multiple times. As the system modules are dependent on each other, the test is executed with a function-able system without executing single methods in an isolated environment. Therefore, module and integration test are conducted simultaneously. Table A.1 in appendix A depicts the single methods and average times of execution of these methods. As a benchmark for the execution time in the Unity3d system, the default physics engine update time of 20ms is used. Values below 20ms allow an event calculation as intended by the engine and enable Unity3d to render a consistence visual simulation representation. The benchmark of the ROS2 process time is the frequency of the sensors data generation e.g. the LIDAR sensor is set to 2 Hertz.

The processor and GPU utilization in the system during the tests is shown in table 4.8. For the CPU, the load in three system states is taken with the tool *htop*. With a total of 16 cores, a value of 16 results in the CPU being full loaded. Values above 16 result in queued processes. The system in idle mode uses 60% of one core and the system with an active Unity3d simulation and running ROS2 nodes uses six cores and 56 % of the seventh core. Regarding the GPU load that is measured with *nvidia-smi*, only 49% is used with Unity3d and ROS2 running. It can be stated, that the CPU and GPU processing power constitute no limitation for conducting tests on the system.

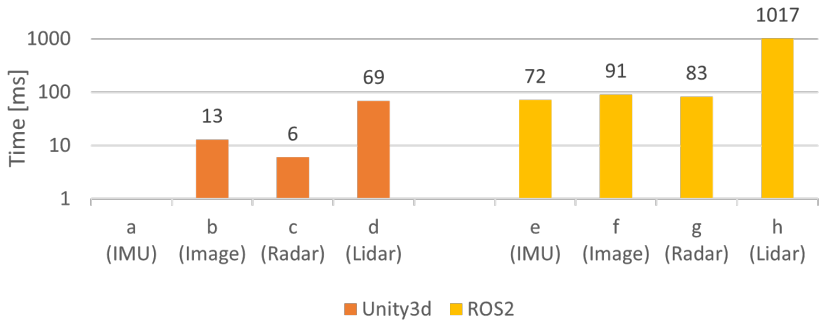


Figure 1.: Methods average execution time (total of 165000 execution)

Table 4.8.: CPU and GPU load

System processes	CPU Load (max 16) $\varnothing 1min$
Idle	0.6
Unity3d	3.91
Unity3d and ROS2	6.56
System processes	**GPU Load (max 100%)**
Unity3d and ROS2	49%

To test the implementation of the modules in the system, the average execution time of the methods for the IMU, Image, RADAR and LIDAR in Unity3d and ROS2 is taken. Figure 1 depicts these values. In the first category with execution times from (a) to (d), the methods for the IMU data (a) equals to zero as the average is below the vertical axis that is logarithmic scaled in milliseconds. (a), (b) and (c) are below the 20ms with which Unity3d updates its physics per default. The calculation of the LIDAR data (d) exceeds 20ms with a value of 69ms. This makes a more detailed analysis necessary to narrow down on the methods that are responsible for the high computation time.

The second category in figure 1 is represented by the ROS2 subsystem. The according values for (e) to (h) represent the time that is required to process and visualize the data of the sensors. With a value of above 72ms, the IMU (e) with an update rate of 50 Hertz exceeds its benchmark value of 20ms values by 52ms. The same counts for the calculation of the LIDAR data (h). With an update rate of 500ms, the processing time surpasses the defined benchmark by 517ms. Both the methods of (e) and (h) are looked upon in greater detail in the following. (f) and (g) can be considered as performing well in comparison to its update rate of 500ms.

Figure 2 provides more detailed information about the execution time of methods in Unity3d with a box and whisker plot. The figure is divided into two plots with

the plot to the left describing the methods execution time in milliseconds up to 18 milliseconds and the plot to the right methods with a longer execution time up to 180 milliseconds. A total of fourteen methods of the categories Radar, Image, IMU and Lidar are investigated over a total of 126000 executions with an update frequency of two Hertz with the IMU being an exception with 50 Hertz. In the plot to the left, the raycast of the RADAR (1b) takes the longest time by comparison of medians. However, with 4 milliseconds this value can be considered as sufficient low compared to the time of 20ms with which the the physics engine of Unity3d calculates new data per default. In terms of reliability, there are outliers which are higher than the 4ms of (1b). This is of importantance in particular as they compromise the reliability of data generation in a fixed time. Especially publishing the IMU data (1i) stands out in terms of difference from the median to the outliers. The longest execution time of (1i) is around 17ms. However, when using the benchmark of the repetition rate of the physics update, this value can be considered as sufficiently fast to avoid missing physics events.

The plot to the left depicts the three most time consuming methods on Unity3d with the data generation of the LIDAR (2a), publishing the data of the LIDAR (2b) and generation i.e. rendering of the image. By exceeding the physics update repetition rate, methods (2a) and (2c) show a very limited range of the times of the outliers. In contrast executing (2c) can take up to 170ms with a median of 13ms. It can be assumed that for the three outliers that are considerably higher than the median an exception occurs e.g. other processes are scheduled to be executed on the same processor core as (2c). To summarize, for a higher performance in Unity3d, (2b) has to be optimized towards a higher overall performance and modifications in the system should lead to avoidance of the peak values of (2c).

For the methods that are executed in ROS2 figure 3 provides two figures with a similar structure to figure 2. The figure to the left plots data up to 300ms and the plot to the right data up to 3s in a box and whisker representation of about 27300 executions. Again the methods that are depicted can be looked up in appendix A in table A.1. In figure 3, (3a) to (3h) process times at below 300ms even for exceptional outliers are shown, which is sufficient for the application at hand. Going into further detail, method (3g) that computes the bounding boxes for the LIDAR data is noteworthy as it shows an comparable high median of just below 200ms. Thus (3g) provides a first method that can be optimized towards a lower computing time to minimize the LIDAR data processing time.

In the plot to the right it becomes clear that the execution times of (4a) to (4b) can exceed 500ms. That is a critical limit as the frequency from the data generation in Unity3d is set to two Hertz. With the methods execution time above 500ms the ROS2 network is not able to process the data in time which can lead to data being ignored. Upon further investigation, it becomes clear that these methods solely visualize data. As the data visualization is implemented with the matplotlib library

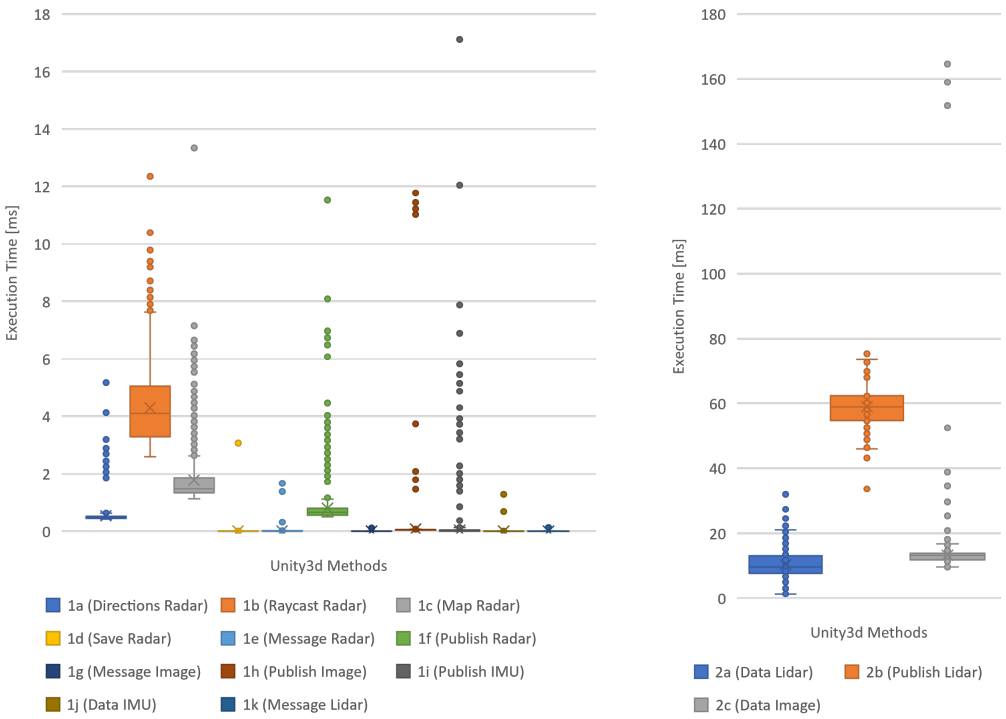


Figure 2.: Unity3d methods time (total of 126000 executions)

for Python, finding another library more suited to plot data in a repeated cycle provides a way to improve the time needed for visualization. A further chance of improvement is given by the visualization of already processed data e.g. visualizing the bounding boxes of the LIDAR data instead of the whole point cloud.

4.3. Part Integration Test

Continuing with the time that is required to transmit the data, as a first step the average size of the messages that are transmitted is taken. To avoid measurements that are delayed by processing data in ROS2, empty nodes with the only purpose of accepting data are written. In table 4.9, (5a) to (5e) show the average size of 100 message of the RADAR, LIDAR, IMU and Image sensor respectively as well as the total transmitted size per second. With 285 KB/s, (5b) stands out of (5a), (5c) and (5d) and makes up a large part of the sum of messages sizes of about 291 KB/s. Considerably larger than the message size is the average intern network load of the system that is measured with the tool *nload* and calculated over 100 samples. 1.62 MB/s as shown in table 4.10 are send over the network under load of which only 24.418 KB/s are used from the operating system. The high value can be explained with the necessity of converting and resending the data between the Unity3d environment in C# and the ROS2 environment in C++ over the Node.js

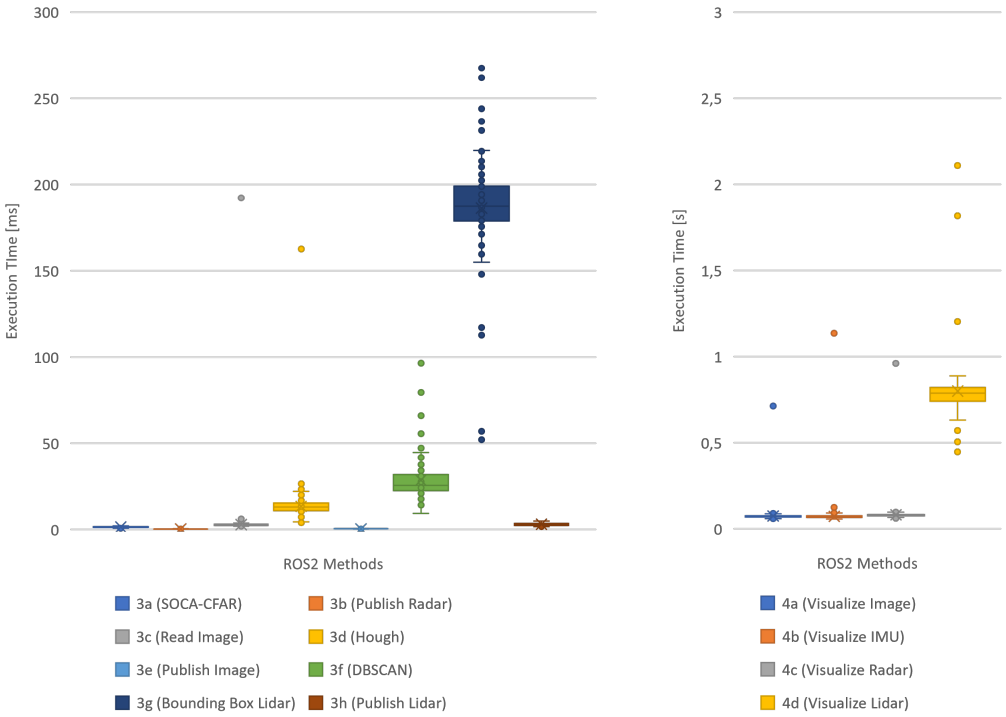


Figure 3.: ROS2 methods time in (total of 27300 executions)

framework in Java. However, a multiplication of about five relative to the total size of the messages is considered as high and can be investigated in further studies to increase the system performance and decrease the latency.

Table 4.9.: Message size in ROS2

Name	Size ∅100
5a (Radar)	5.78 KB/s
5b (Lidar)	285.37 KB/s
5c (IMU)	32 B/s
5d (Image)	88 B/s
5e Sum	291.270 KB/s

Table 4.10.: Network Load

Network Load	∅300s
Idle In/Out	24.418 KB/s
Publishing In/Out	1.62 MB/s

Investigating the latency, to the left of figure 4, (i) to (l) provide the values for the average transmission time from the Unity3d system to the ROS2 system. With an average of 86ms, the LIDAR data (h) stands out from the other values but can be considered as acceptable as the LIDAR data includes a high amount of single point

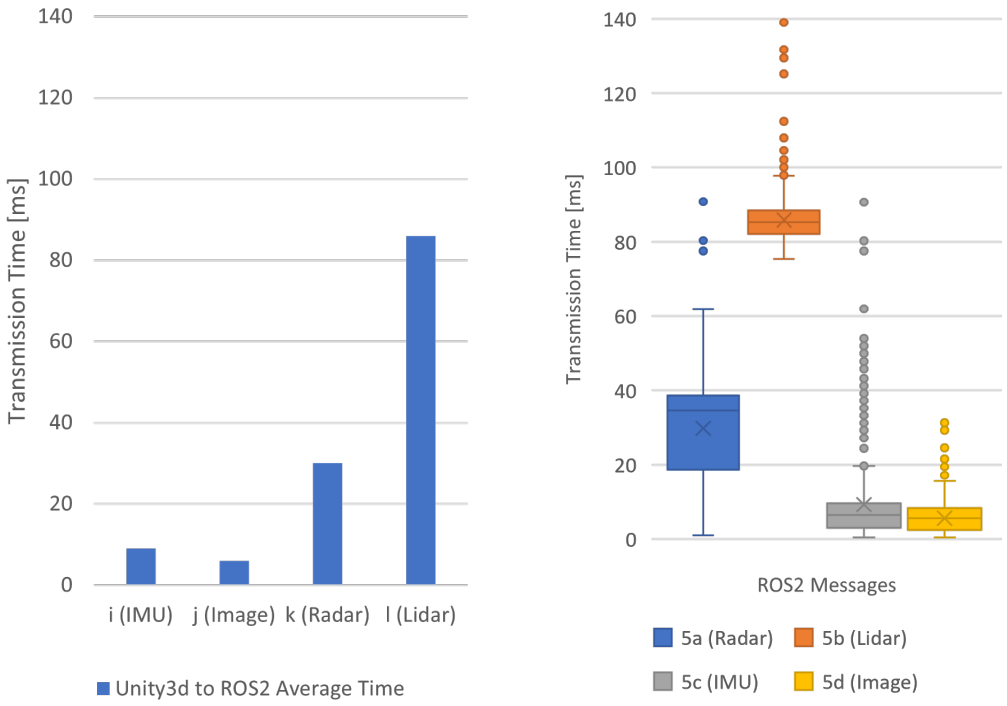


Figure 4.: ROS2 methods time in (total of 12600 executions)

values that have to be transmitted. However, more important than the average time is the reliability with which the transmission time is in a certain interval of time. Therefore, a more detailed representation of the data is shown in figure 4 to the right with a box and whiskers plot. It becomes clear that the average transmission time of the message conceal a highly scattered latency. Especially for the IMU (5c) with only 32 B/s of data size, transmission times of above 80ms are possible with an median of below 10ms. Similar to (5c), the difference of the median to the highest latency value measured in (5b) is about 50ms. However, no requirement on the real time behaviour is set for the system and times of below 140ms are within a range that can be considered as sufficient for an adequate response in the scope of the appliance at hand. Detailing the real time definition of the appliance may be considered in further studies.

4.4. Validation Software

In this section, the implemented system is validated by using the proposed system architecture, the system specification and requirements defined in terms of safety. Central to the implementation is the software framework ROS2 that can be used to connect real sensors as well as virtual sensors, control actuators and share information with a control center. The library that is chosen to connect to the virtual

sensors allows for a physical separation of the processing units that calculates the virtual sensors data and the subsequent processing of the data. Unity3d as the simulation environment proofed to be able to generate virtual sensor data and it is possible to resemble different types of sensors to percept the virtual environment. Therefore the chosen modules and the implemented system architecture is in compliance with the specified design. The system implementation is oriented at the V-Model and therefore complies with the demand to use a valid software safety life cycle model. Verifying the software further, the part of the regulations that relate to the design of a software framework state, that the information that result in any autonomous decision making is to be shared with the personell of the MASS trial. The software as implemented fullfills this demand by visualizing the processed data of the sensors after each processing step.

Continuing with a more detailed analysis, the system as implemented is tested against the criteria set in section 7.4 of VDE 0803-3 [33] and described in section 3.1. A total of 14 criteria should be met in the software design and development process. In table 4.11, the criteria of the test that is conducted is given with the result being positive or negative. To start with the first two tests that are specified, it can be concluded that there no fault in the concept is detectable (1) and the overall system is designed with a focus on easy comprehensibility (2). With the utilization of ROS2 as a central framework, a deterministic behaviour of the data processing can be ensured (3). As only the Unity3d environment is used in the system and therefore influences of a real environment are missing, criteria (4) can not be evaluated. Both Unity3d and ROS2 provide possibilities to ad or replace components where required through the asset store in the first case and packages in the latter. This option results in a modular (5) and changeable code (6). By making use of classes and methods, the code in Unity3d is divided into several submodules that are easily testable. However, the methods of one class are dependent on each other. Therefore testing each method for itself requires a considerable amount of additional code that simulate the respective methods before the method that is to be tested. The same is the case for the code in ROS2. With a division into nodes, classes and methods, methods are still dependent on each other. However, as the separate testing is not a requirement, criteria (7) and (8) are considered to be met. By making use of a structured process oriented at the V-model throughout every aspect of the system design is documented which includes the intended design of the system (9), the integration test (10) that falls together with the module test (11) and the safety validation. In the module implementation, a summary is provided by every class and method that is written to ease the understanding of the purpose of the code (13). The author of the code is documented, too (14). It is noted that no liability or responsibility is taken for the code.

Table 4.11.: Safety criteria in system validation

Criteria	Criteria met
(1) Without faults in the concept	✓
(2) Easily comprehensible	✓
(3) Display a deterministic behaviour	✓
(4) Fault tolerant in the event of external events	✗[1]
(5) Modularity	✓
(6) Changeable code	✓
(7) Divided into several more detailed submodules	✓[2]
(8) Testable	✓
(9) Intended design of the system documented	✓
(10) Integration Test documented	✓
(11) Module test documented	✓
(12) Safety validation of the system is documented	✓
(13) Software is documented	✓
(14) Responsibility is documented	✓[3]

[1] no test specified, only the simulation is used as a test environment

[2] submodules are dependent on each other, separate testing requires more effort

[3] translates to the documentation of the author of the code

CHAPTER 5

Conclusion and Outlook

5.1. Conclusion

In this thesis a software architecture for virtual sensors in autonomous small vessels is conceptualized and implemented that allows for the integration of virtual and physical sensors. A structured and safe design process is obtained by following the V-model for software designs. To avoid possible restrictions imposed by the limitations of the chosen software frameworks, a focus is laid on the system specification. With ROS2, Unity3d and Node.js a solution is specified which not only meets the requirements but opens up the opportunity for further implementations that expand the scope of the introduced system. In conjunction with the system design, an Unity3d scene is created that resembles the topographic as well as visual data from an existing location. Furthermore, a viable way is shown to reconstruct and implement a RADAR in a virtual environment. By integrating four different virtual sensors into the scene, the scalability of the design is demonstrated. It is shown, that algorithms in ROS2 which comprise a cluster analysis (DBSCAN) and a false alarm detector (SOCA-CFAR) are able to process the environmental data of the sensors and thus functions are implemented that are vital for the navigation subsystem of a GNC system. Furthermore, routines that calculate bounding boxes around clustered LIDAR data provides condensed environmental data and a way is shown to connect to the guidance subsystem of a GNC system with a low data rate.

In compliance with the defined requirements, the final result of this thesis is an architecture that allows for the easy incorporation of software that is vital to the design of autonomous inland navigation vessels. In addition, a variety of tested algorithms demonstrate the diverse opportunities that result of the chosen solution.

5.2. Suggestions for Future Work

Further waypoints of detailing the proposed architecture are to be found in the specified functions of a GNC system. Starting with the implementation of a guidance subsystem that utilizes the information of the guidance subsystem, also a control subsystem has to be realized that connects to the actors of a vessel. It remains open which algorithms prove to full fill the required functionality of this subsystems. In this context, modelling the virtual scene in greater detail is clearly beneficial to the quality of information that can be derived from the simulation and exported to the behaviour of a real vessel. Especially adding a more sophisticated simulation of water is expected to increase the possibilities of testing algorithms closer to conditions that may occur in a real world scenario. With the architecture providing an easy way to embed real sensors, it remains open how the system reacts to a substitution of the virtual sensors and influences of a real environment.

With the focus on demonstrating the versatility of the architecture, the focus in the implementation is on a variety of use cases instead of optimizations. Therefore, concerning the realized software, a number of possible improvements show up e.g. improving the GPU utilization in Unity3d. The interested reader may refer to appendix B where possible improvements to the existing software are detailed.

Bibliography

[1] S.P.A. Alewijnse, T.M. Bagautdinov, and M. de Berg. "Progressive Geometric Algorithms". In: *Journal Of Computational Geometry* 6.2 (2015), pp. 72–92.

[2] Aptiv. *Aptiv-ESR-25*. Electronic Article. 2020. URL: https://hexagondownloads.blob.core.windows.net/public/AutonomouStuff/wp-content/uploads/2020/10/aptiv-esr-25-24v-datasheet.pdf (visited on 07/28/2021).

[3] Xsens Technologies B.V. *MTi 600-series*. Electronic Article. 2020. URL: https://cdn2.hubspot.net/hubfs/3446270/Downloads/Leaflets/MTi%20600-series%20Datasheet.pdf?__hstc=&__hssc=&hsCtaTracking=2c806b91-154c-4c28-9935-4ee210077f32%7C567a6d03-3dc0-4249-8f49-df9ea7f7a49d (visited on 07/28/2021).

[4] Michael R. Benjamin et al. "Nested autonomy for unmanned marine vehicles with MOOS-IvP". In: *Journal of Field Robotics* 27.6 (2010), pp. 834–875.

[5] Xin Bin. *Environmental Perception Technology for Unmanned Systems*. Vol. 1. Huazhong University of Science: Springer, Singapore, 2021.

[6] Marius Brinkmann et al. *Learning from Automotive: Testing Maritime Assistance Systems up to Autonomous Vessels*. IEEE, 2017.

[7] Yanchun Zhang Yu-Chen Song Hai-Dong Meng. "Clustering analysis and its applications". In: *2010 Second IITA International Conference on Geoscience and Remote Sensing* 1 (2010), pp. 514–517.

[8] Nvidia Corporation. *Jetson AGX Xavier Development Kit*. Electronic Article. 2018. URL: https://developer.download.nvidia.com/embedded/L4T/r32-3-1_Release_v1.0/jetson_agx_xavier_developer_kit_user_guide.pdf?sFjjAKgzfRvtad9LkUauVosSs5P2xq5bVpOWwmBviH19hKAcrdAUKIqn0WyUcdNhLIUGug7_meQG3OPiBeeTvoXkByA63_2U207hf3DNZHrJcotDBotN2NULHq4db7ljLQYgIu2J_uxKqEC9vWS8tAHeNbNPwMWGoCCnZWOUJcAfbh_G6mSgXTLSfxJ_p1M (visited on 07/28/2021).

[9] "Quaternions". In: *Kinematic Analysis of Robot Manipulators*. Ed. by I. I. I. Carl D. Crane and Joseph Duffy. Cambridge: Cambridge University Press, 1998, pp. 381–399.

[10] DFRobot. *A01NYUB Waterproof Ultrasonic Sensor*. Electronic Article. 2021. URL: https://wiki.dfrobot.com/A01NYUB%20Waterproof%20Ultrasonic%20Sensor%20SKU:%20SEN0313 (visited on 07/28/2021).

[11] K. D. Do, Z. P. Jiang, and J. Pan. "Robust adaptive path following of underactuated ships". In: *Automatica* 40.6 (2004), pp. 929–944.

[12] A. Driewer. "Modellierung von 3D-Time-of-Flight-Sensoren und -Systemen". PhD-Thesis. University of Duisburg-Essen, 2016.

[13] T. Erez, Y. Tassa, and E. Todorov. "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4397–4404.

[14] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, pp. 226–231.

[15] Hans-Josef Fischer. *Digital Terrain Model*. Web Page. URL: https://www.opengeodata.nrw.de/produkte/geobasis/hm/dgm1_xyz/dgm1_xyz/ (visited on 07/28/2021).

[16] Leila De Floriani and Paola Magillo. "Triangulated Irregular Network". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer ÖZsu. Boston, MA: Springer US, 2009, pp. 3178–3179.

[17] Jonah Gamba. *Radar Signal Processing for Autonomous Driving*. Electronic Book. 10.1007/978-981-13-9193-4. 2020.

[18] G. Lufft Mess- und Regeltechnik GmbH. *WSx-UMB*. Electronic Article. 2020. URL: https://www.lufft.com/download/manual-lufft-wsxxx-weather-sensor-en/ (visited on 07/28/2021).

[19] Carlos San Vicente Gutiérrez et al. "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications". In: *arXiv preprint arXiv:1809.02595* (2018).

[20] Axel Hahn et al. "Requirements for e-Navigation Architectures". In: *International Journal of e-Navigation and Maritime Economy* 5 (2016), pp. 1–20.

[21] D. Cáceres Hernández, V. Hoang, and K. Jo. "Lane Surface Identification Based on Reflectance using Laser Range Finder". In: *2014 IEEE/SICE International Symposium on System Integration*, pp. 621–625.

[22] Ahmed Hussein, Fernando Garcia, and Cristina Olaverri Monreal. "ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation". In: *2018 IEEE International Conference on Vehicular Electronics and Safety, ICVES*.

[23] J. Illingworth and J. Kittler. "A survey of the hough transform". In: *Computer Vision, Graphics, and Image Processing* 44.1 (1988), pp. 87–116.

[24] FLIR Integrated Imaging Solutions Inc. *BFS-U3-13Y3*. Electronic Article. 2017. URL: https://flir.app.boxcn.net/s/4iegv21td532aowlt4wzhq4jzafojxrt (visited on 07/28/2021).

[25] Ali Karimian et al. "Refractivity estimation from sea clutter: An invited review". In: *Radio Science* 46.6 (2011).

[26] S. Kato et al. "An Open Approach to Autonomous Vehicles". In: *IEEE Micro* 35 (2015), pp. 60–68.

[27] S. Kato et al. "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems". In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296.

[28] PHOENIX CONTACT GmbH Co. KG. *UM EN FL WLAN 1100*. Electronic Article. 2018. URL: https://www.phoenixcontact.com/online/portal/de?uri=pxc-oc-itemdetail:pid=2702534&library=dede&tab=1 (visited on 07/28/2021).

[29] Oussama Khatib. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". In: *The International Journal of Robotics Research* 5.1 (1986), pp. 90–98.

[30] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3, 2149–2154 vol.3.

[31] Anis Koubaa. *Robot Operating System (ROS) : The Complete Reference (Volume 4)*. 1st ed. 2020. Studies in Computational Intelligence. [HerausgeberIn]. Cham: Springer International Publishing Imprint: Springer, 2020.

[32] Holger Lange. *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements (IEC 61508-1:2010)*. Online Database. Feb. 2011.

[33] Holger Lange. *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements (IEC 61508-3:2010)*. Online Database. Feb. 2011.

[34] J. D. Lawrence. "An overview of software safety standards". In: Research Org.: Lawrence Livermore National Lab., CA (United States), Sponsor Org.: Nuclear Regulatory Commission, Washington, DC (United States).

[35] Velodyne Lidar. *VLP-16*. Electronic Article. 2018. URL: https://velodynelidar.com/products/puck/#downloads (visited on 07/28/2021).

[36] Jialun Liu et al. "Literature review on evaluation and prediction methods of inland vessel manoeuvrability". In: *Ocean Engineering* 106 (2015), pp. 458–471.

[37] Zhixiang Liu et al. "Unmanned surface vehicles: An overview of developments and challenges". In: *Annual Reviews in Control* 41 (2016), pp. 71–93.

[38] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. “Exploring the performance of ROS2”. In: IEEE EMSOFT, pp. 1–10.

[39] *Masterplan Binnenschifffahrt*. Report. Bundesministerium für Verkehr und digitale Infrastruktur, 2019.

[40] International Maritime Organization. *Interim Guidelines for MASS Trials*. Electronic Article. 2019. (Visited on 08/04/2021).

[41] G. Peeters et al. “An unmanned inland cargo vessel: Design, build, and experiments”. In: *Ocean Engineering* 201 (2020).

[42] Wilfried Plaßmann. “Funkmesstechnik”. In: *Handbuch Elektrotechnik*. 2016. Chap. Chapter 104, pp. 1277–1281.

[43] Stefan Profanter et al. *OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols*. 2019.

[44] Morgan Quigley et al. *ROS: an open-source Robot Operating System*. Vol. 3. 2009.

[45] Strenge Rainer et al. “Driving Assistance Systems for Inland Vessels based on High Precision DGNSS”. In: *PIANC-World Congress Panama City*.

[46] H. Rohling. “Radar CFAR Thresholding in Clutter and Multiple Target Situations”. In: *IEEE Transactions on Aerospace and Electronic Systems* AES-19 (1983), pp. 608–621.

[47] Erich Schubert et al. “DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN”. In: *ACM Trans. Database Syst.* 42.3 (2017), Article 19.

[48] A. Spieckermann. “Photodetektoren und Auslesekonzepte für 3D-Time-of-Flight-Bildsensoren in 0,35 um-Standard-CMOS-Technologie”. PhD-Thesis. University of Duisburg-Essen, 2010.

[49] International Organization for Standardization. *ISO 17894:2005(en): Ships and marine technology — Computer applications — General principles for the development and use of programmable electronic systems in marine applications*. Online Database. Feb. 2005.

[50] Yuliang Sun, Tai Fei, and Nils Pohl. “A High-Resolution Framework for Range-Doppler Frequency Estimation in Automotive Radar Systems”. In: *IEEE Sensors Journal* 19.23 (2019), pp. 11346–11358.

[51] Christa Sys et al. “Case Studies on Transport Policy”. In: *World Conference on Transport Research Society* (2020).

[52] Hunts-berger Terrance et al. *Control Architecture for Robotic Agent Command and Sensing*. Electronic Article. 2008. URL: `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080048021.pdf` (visited on 08/04/2021).

[53] *Transport 2050*. Electronic Article. 2011. URL: https://ec.europa.eu/commission/presscorner/detail/en/IP_11_372 (visited on 07/28/2021).

[54] European Commission Mobility and Transport. *EU Operational Guidelines for Safe and Sustainable Trials of Maritime Autonomous Surface Ships*. Electronic Article. 2020. (Visited on 08/04/2021).

[55] Yiqiu Wang, Yan Gu, and Julian Shun. "Theoretically-Efficient and Practical Parallel DBSCAN". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, pp. 2555–2571.

[56] *Watertruck+*. Web Page. 2020. URL: http://www.watertruckplus.eu/ (visited on 07/28/2021).

[57] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: an efficient data clustering method for very large databases". In: *SIGMOD Rec.* 25.2 (1996), pp. 103–114.

[58] Z. Zhou et al. "Simulation Platform for USV Path Planning based on Unity3D and A* Algorithm". In: *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, pp. 1–6.

[59] Tomasz Zubowicz et al. "Marine autonomous surface ship - control system configuration". In: *IFAC* 52.8 (2019), pp. 409–415.

List of Figures

List of Tables

APPENDIX A

Execution time of the systems methods

Table A.1 details the execution time on average of every method that is used in this thesis. In the first column, an abbreviation for the method is defined. Followed by the class the method can be found in, the methods name is given. The name is followed by the average execution time of the according message. The last column shows the average execution time of all methods of one class. Methods (1) to (2) belong to the Unity3d environment and methods (3) to (4) relate to the ROS system. Every horizontal single line represents a different sensor and every horizontal double line indicates a different system. An exception from the previous explanation is made for the case (5a) to (5d). These are the messages for a ROS interface and the average execution time translates to the time that is needed for a transmission from Unity3d to ROS. As the messages are transmitted in parallel to each other, they are not summed up.

Table A.1.: Average execution time of methods

Abbreviation	Class/Package	Method/Message	∅ [ms]	Sum [ms]
1a (Directions Radar)	RadarSensor	Calculate direc.()	1	6
1b (Raycast Radar)	RadarSensor	CommandRaycast	4	
1c (Map Radar)	RadarSensor	MapToPlane()	2	
1d (Save Radar)	RadarSensor	Save()	0	
1e (Message Radar)	ROSPublishRadar1	SetMessage()	0	
1f (Publish Radar)	ROSPublishRadar1	Publish()	1	
2c (Data Image)	SensorImage	UpdateImage()	13	13
1g (Message Image)	ROSPublishImg	SetMessage()	0	
1h (Publish Image)	ROSPublishImg	Publish()	0	
1i (Publish IMU)	ROSPublishIMU	Publish()	0	0
1j (Data IMU)	SensorIMU	GetIMUData()	0	
1k (Message Lidar)	ROSPublishXYZ	SetMessage	0	69
2a (Data Lidar)	LidarSensor	Lidar()	10	
2b (Publish Lidar)	ROSPublishXYZ	Publish()	59	
5a (Radar)	gnc/interfaces	/radar1/img	30	30
5b (Lidar)	gnc/interfaces	/lidar1/xyz	86	86
5c (IMU)	gnc/interfaces	/imu1/accvel	9	9
5d (Image)	gnc/interfaces	/cam1/pixel	6	6
3a (SOCA-CFAR)	get_radar	bounding_box()	2	83
3b (Publish Radar)	get_radar	publish()	0	
4c (Visualize Radar)	get_radar	visualize_cluster()	81	
3c (Read Image)	get_image	read_image()	3	91
3d (Hough)	get_image	lines_extraction()	13	
3e (Publish Image)	get_image	publish_lines()	1	
4a (Visualize Image)	get_image	visualization()	74	
3f (DBSCAN)	get_lidar	dbscan()	29	1017
3g (B. Box Lidar)	get_lidar	bounding_box()	186	
3h (Publish Lidar)	get_lidar	publish()	3	
4d (Visualize Lidar)	get_lidar	visualize_cluster()	799	
4b (Visualize IMU)	get_imu	visualize_data()	72	72

APPENDIX B

Improvements to the existing system

In the following, a list of implementations is shown that may improve the software of the system. In itself the single points are expected to contribute only little to the overall system performance but may increase the performance considerably when summed up.

- The GPU utilization in the Unity3d scene is low even with a low framerate due to a high computational effort. Parallelizing more tasks would contribute to a higher framerate as the computation load is focused not only on one GPU unit.
- The intern network load of the operating system is considerably higher than the messages that are send from Unity3d. The image message as implemented in this work provides an example of dividing a message in metadata that is send over the ROS network and data that is saved by the operating system and captured by ROS upon request. However it has to be kept in mind that this approach is not in compliance with the design rules for a ROS package.
- As shown, visualizing high amounts of data with the python library *matplotlib* takes a long time when executed in the ROS environment. Finding another way to plot data with *matplotlib* or find an optimized library for this purpose may prove to be advantageous.
- The calculation of the bounding boxes of the LIDAR cluster takes a long time due to recurrent iterations over the dataset. Elaborate on the used algorithm can result in lower computation time.
- Due to the raycasts in the simulation that reconstruct the LIDAR laser beams, the resulting virtual sensor takes a long time until all rays are send. Dividing

the raycast into beams that are independent from each other could provide an opportunity to utilize multiple cores in addition to the *JobHandle* that is already used. Investigating in other steps that need a long time to be executed may result in further possibilities of optimization.

- As the methods used in this work are dependent on each other, they were not tested in an isolated environment. Creating data that simulates the output of methods, single routines can be tested and influences, e.g. queing data, of other methods can be avoided.

- The rendering of the reflection on water requires a separate camera in the Unity3s scene. Several water planes are used and thus the computational effort is rising. Avoiding a multitude of rendered cameras can increase the frames per second with which events are calculated in Unity3d. As proposed, using a more elaborated water simulation can solve this problem.

- To simulate the analogous beam signal of a real RADAR, many ray casts are necessary in Unity3d. Using a box raycast that is given by the physics engine of Unity3d, instead of many rays, one cast over an area is made with a three dimensional shape. The advantage is the reduced computational effort due to the reduced ray casts.

- Instead of using a decentralised solution in which every data is saved locally in the classes that transmit data to other classes, a central location to can be allocated from where the classes do queries for the data they need. This avoids the recurrent allocation of memory for the data e.g. from sensors.